Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2013

João Cardoso

DETIboot: distribuição e arranque de sistemas Linux com redes WiFi

João Cardoso

DETIboot: distribuição e arranque de sistemas Linux com redes WiFi

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de José Vieira e André Zúquete, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Tomás António Mendes Oliveira e Silva
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Professor Doutor José Legatheaux Martins
Professor Catedrático da Universidade Nova de Lisboa (arguente)

Professor Doutor José Manuel Neto Vieira
Professor Auxiliar da Universidade de Aveiro (orientador)

Professor Doutor André Ventura da Cruz Marnoto Zúquete
Professor Auxiliar da Universidade de Aveiro (co-orientador)

agradecimentos / acknowledgements

Agradeço a todo o corpo docente das disciplinas lecionadas ao longo do meu percurso académico, principalmente aos meus orientadores, Professor André Zúquete e Professor José Vieira, por todo o conhecimento transmitido, essencial para a realização desta dissertação.

Agradeço também à família e amigos mais chegados, por todo o apoio recebido e momentos vividos, e por último, a todos os colegas de curso com quem tive a oportunidade de trabalhar e adquirir conhecimentos suplementares aos lecionados durante o meu percurso académico.

Resumo

Com a evolução da tecnologia nos últimos anos, os computadores portáteis tornaram-se capazes de satisfazer grande parte das necessidades dos utilizadores a nível pessoal, proporcionando uma mobilidade que os computadores de secretária não permitiam. Assim, hoje em dia, os computadores pessoais são cada vez mais portáteis, em detrimento dos computadores fixos.

É comum as universidades terem computadores fixos disponíveis para os estudantes nos seus departamentos, mas devido à mudança de contexto dos computadores pessoais, a mobilidade destes deixou de ser um problema o que leva a que os alunos optem muitas vezes por utilizar os seus computadores portáteis em detrimento dos computadores fixos disponibilizados pelas universidades, tornado-se assim estes últimos obsoletos.

A manutenção dos computadores fixos disponibilizados pelas universidades é bastante dispendiosa e apesar de pouco utilizados, existem alguns aspetos em que a utilização dos computadores pessoais dos alunos não é desejável, em particular a realização de exames práticos onde é necessário restringir as permissões dos utilizadores. Nos computadores fixos das universidades, os docentes têm privilégios de acesso sobre o sistema operativo, permitindo assim manipular as permissões dos restantes utilizadores, ao contrário dos computadores pessoais dos alunos, onde é o próprio aluno que tem total controlo sobre as mesmas no seu sistema operativo.

O DETIboot vem resolver o problema de utilização de computadores pessoais durante a realização de exames práticos, oferecendo aos docentes a oportunidade de executar facilmente um sistema operativo, configurado pelos próprios, nos computadores portáteis dos alunos, passando assim o controlo das permissões dos utilizadores para os docentes.

Para isso, foi desenvolvido um método de transferência de ficheiros que consiste na transmissão em difusão sem fios, em modo ad-hoc, baseado em códigos Fountain de forma a suprimir a perda de pacotes da comunicação sem fios. Com isto, foi também desenvolvido um método de arranque do sistema operativo que utiliza uma imagem transmitida como sendo a raiz do mesmo, em detrimento de um dispositivo de armazenamento como é habitual.

Abstract

With the improvements of computers technology in the last years, laptop computers have become powerful enough to satisfy the needs of the users in personal context with the mobility that are not present in desktops. So, nowadays, personal computers are mostly laptops instead of desktop computers.

Universities usually have desktop computers available to the students in campus but with the mobility present in today laptops, many students prefer to use their own laptops in detriment of University desktops, making this computers useless.

Universities spend many resources in maintenance of these computers and beside they became useless, in some cases the use of students laptops is not desirable, like laboratory exams where is necessary restrict users permissions. In university desktops, professors have privileged access over the operating system, giving them the possibility of change users permissions, unlike students laptops where themselves have full control over the operating system.

DETIboot comes to resolve the problem of using personal computers on pratical exames, by giving professors the opportunity of easily run a custom operating system, configured by their own, in students laptops, passing on control of users permissions to the professors.

For this propose, was developed a file transfer method consisting on wireless broadcast transmission, in ad-hoc mode, based on Fountain Codes to suppress packets loss of wireless communication. With this, was also developed a boot operating system method using transmitted file as being the root, rather than a store device as usually.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Esta dissertação surgiu de uma necessidade atual das universidades, mais concretamente em relação ao desuso, por parte dos alunos, dos computadores fixos disponibilizados pelas mesmas. Existe ainda a necessidade de manter estes computadores nas universidades, acarretando custos elevados, devido a algumas tarefas não realizáveis nos computadores pessoais dos alunos. Assim, o objetivo principal desta dissertação visou a criação de mecanismos que permitam tornar tais tarefas realizáveis nos computadores pessoais dos alunos. Em particular, estudou-se um processo rápido e flexível de carregamento de um sistema operativo Linux, numa população de máquinas arbitrariamente grande durante um período de tempo curto.

No final desta dissertação, o objetivo principal foi cumprido com resultados bastante satisfatórios (ver capítulo 5), para além de ainda ser possível utilizar a solução encontrada noutros contextos, como por exemplo a realização de *workshops*, ou qualquer outro contexto em que seja favorável a distribuição de um sistema operativo pré configurado.

1.1 Motivação

Hoje em dia, é indispensável a qualquer aluno universitário ter um computador pessoal, de forma a poder realizar as mais diversas tarefas ao longo do seu percurso académico. Com a evolução da tecnologia, os computadores portáteis tornaram-se suficientemente poderosos para desempenhar essas tarefas, para além de poderem estar sempre presentes no trabalho diário e algo nómada dos alunos, ao contrário dos computadores fixos, que embora consigam ter normalmente melhor performance, carecem de falta de mobilidade. Assim, grande parte dos estudantes universitários possui um computador portátil e muitos preferem transportá-lo e prescindir dos computadores fixos disponibilizados pelas universidades.

Os computadores disponibilizados pelas universidades são bastante dispendiosos, tanto em termos de custo material como a nível de recursos humanos, e com o aumento do seu desuso justifica-se diminuir estes custos. No entanto, existem algumas situações em que a utilização dos computadores pessoais dos alunos não é desejável, como a realização de exames práticos, visto que os alunos têm controlo total sobre os seus computadores, ao contrário dos computadores fixos das universidades, aos quais normalmente apenas os docentes têm acesso privilegiado, podendo assim configurar aplicações e definir as permissões de acesso dos restantes utilizadores, limitando deste modo a utilização dos alunos.

Assim, a motivação principal para esta dissertação foi a de fornecer alternativas viáveis aos computadores fixos disponibilizados pelas universidades, permitindo assim que estas possam

reduzir ao máximo o número de computadores disponibilizados e diminuir substancialmente ou eliminar os custos necessários para a manutenção destes.

1.2 Problema

Durante a realização de exames práticos em que é necessária a utilização de sistemas computacionais, muitas vezes é necessário restringir as permissões destes, como o acesso à Internet ou a dispositivos de armazenamento de dados (internos ou externos), para que os alunos não tenham acesso a informação não autorizada ou ajuda externa personalizada que os possa beneficiar durante a realização da prova.

Utilizando os computadores fixos disponibilizados pelas universidades, os docentes podem configurar os seus sistemas operativos de forma a restringir o acesso a serviços indesejáveis aos alunos, devido ao acesso privilegiado que os docentes têm sobre o sistema operativo dos computadores das universidades.

Nos computadores pessoais dos alunos acontece precisamente o contrário, pois são os alunos que têm acesso privilegiado ao seu sistema operativo, tornando assim difícil, senão impossível, a tarefa dos docentes em restringir as permissões dos alunos nos seus próprios sistemas.

Assim, para que os docentes consigam ter controlo sobre os computadores pessoais dos alunos, é necessário que estes últimos não tenham acesso privilegiado ao sistema operativo, ou seja, não podem ser administradores do sistema operativo a correr nos seus próprios computadores. Mais ainda, os alunos não deverão ter acesso aos sistemas em avanço, de forma a não poderem estudar e contrariar, em antecipação, as suas políticas restritivas.

Com isto, facilmente se identifica o problema em causa, sendo este a necessidade de carregar e executar um sistema operativo pré configurado em computadores de terceiros, permitindo assim aos docentes configurarem um determinado sistema operativo com as restrições e aplicações desejadas e executá-lo nos sistemas computacionais dos alunos, removendo assim o papel de administradores do sistema operativo aos alunos.

No entanto, é necessário que o carregamento do sistema operativo seja efetuado em tempo útil, por população numerosa, e garantir também a deteção de máquinas virtuais de forma a poder invalidar a execução do sistema operativo nas mesmas. Este último aspeto, embora tenha sido estudado e até desenvolvido um método de deteção de máquinas virtuais baseado na instrução máquina RDTSC, não é abordado nesta dissertação, o que não o exclui de um trabalho futuro.

1.3 Contribuição

Nesta dissertação de mestrado é estudado um método de distribuição e execução de um sistema operativo, previamente configurado, de forma a permitir realizar uma tarefa e, simultaneamente, limitando a utilização dos sistemas computacionais de terceiros.

De modo a encontrar a melhor forma de distribuir e executar um sistema operativo em vários sistemas computacionais, foram estudados os diversos métodos de arranque atualmente existentes, bem como todo o processo de inicialização dos sistemas operativos Linux e as características dos sistemas de ficheiros mais utilizados, permitindo assim identificar quais as limitações atuais e quais as tecnologias já existentes, de forma a idealizar a melhor solução possível, para os dias de hoje.

Com este estudo foi então possível identificar que, embora já existam métodos de arranque de sistemas operativos pré configurados (distribuições *live*, ver secção 2.2.1) e até a possibilidade de distribuir o sistema operativo remotamente (PXE, ver secção 2.2.1), ainda existem aspetos que podem ser explorados, como a utilização de redes sem fios de modo a melhorar a acessibilidade por parte dos utilizadores, bem como a utilização de transmissão em difusão, permitindo ter melhor desempenho devido a tornar a velocidade de transferência de dados independente do número de utilizadores.

Posto isto, nesta dissertação foi então estudada uma solução para a distribuição e arranque de sistemas operativos em computadores de terceiros, utilizando redes sem fios WiFi, em modo ad-hoc, como canal de comunicação e utilizando transmissão em difusão, de forma a permitir distribuir e arrancar um sistema operativo num determinado espaço geográfico sem quaisquer restrições quanto ao número de utilizadores, ou seja, escalável.

Assim, para a executar o método de arranque DETIboot, é necessário que exista um nó na rede responsável pela transmissão em difusão do sistema operativo, e para que os restantes nós da rede consigam receber o sistema operativo completo e sem erros, a transmissão é codificada utilizando técnicas de *network coding* (códigos Fountain, ver secção 2.5) de forma a permitir diferentes instantes de inicialização e a suprimir as eventuais perdas de pacotes, sem a necessidade de registo ou *feedback* por parte das máquinas clientes, retirando assim a sobrecarga criada pelo envio de mensagens de notificação. Sem esta sobrecarga, o método de arranque DETIboot torna-se escalável, mantendo um desempenho constante e independente quanto à variação do número de máquinas clientes presentes em simultâneo.

O método de arranque DETIboot foi desenhado para correr nos sistemas computacionais mais comuns entre os alunos, ou seja, computadores portáteis comuns de 64 *bits*. Assim, a solução desenvolvida no âmbito desta dissertação não abrange todos os tipos de sistemas computacionais utilizados pelos alunos, mais concretamente sistemas computacionais *Apple* e máquinas de 32 *bits*. Apesar da existência desta limitação na solução desenvolvida, é possível adaptar esta de forma a obter diferentes versões do método de arranque DETIboot, para diferentes tipos de sistemas computacionais, resolvendo assim a limitação existente.

Capítulo 2

Contexto

Neste capítulo são contextualizados os principais temas abordados no desenvolvimento desta dissertação, facilitando assim a compreensão global da mesma.

Devido ao contexto em que esta dissertação está inserida, arranque e distribuição em redes sem fios de sistemas operativos pré configurados, são abordados temas relacionados com sistemas operativos Linux, redes de comunicação sem fios e técnicas de *network coding*. Assim, no âmbito desta dissertação, foram explorados conceitos relativamente a sistemas de ficheiros (ver secção 2.1), métodos de arranque existentes em sistemas Linux (ver secção 2.2), mudança de núcleo corrente (ver secção 2.3), redes sem fios WiFi (ver secção 2.4) e códigos Fountain (ver secção 2.5).

2.1 Sistemas de ficheiros

Em informática, o sistema de ficheiros é a forma de organização dos dados em dispositivos de armazenamento. Sabendo interpretar o sistema de ficheiros de um determinado dispositivo de armazenamento, o sistema operativo consegue descodificar os dados armazenados no mesmo e realizar operações de leitura e escrita.

Um sistema de ficheiros é assim uma forma de criar uma estrutura lógica de acesso a dados numa partição. Sendo assim, também é importante referir que só pode existir um tipo de sistemas de ficheiros por partição.

Existem vários formatos de sistemas de ficheiros, com características diferentes e diversos fins. Nesta dissertação serão aprofundados 2 sistemas de ficheiros, menos conhecidos, mas importantes no contexto da mesma, Initramfs (ver secção 2.1.1) e Squashfs (ver secção 2.1.2) [1] [2] [3].

2.1.1 Initramfs

O Initramfs é um sistema de ficheiros, usado normalmente no arranque de sistema operativos Linux, que corre unicamente em memória RAM e tem como principal finalidade preparar, montar e inicializar o sistema de ficheiros real do sistema operativo.

Este sistema de ficheiros é um arquivo cpio [4], ou seja, um arquivo de múltiplos ficheiros não comprimido. No entanto é possível (e recomendado) comprimir este arquivo, podendo ser utilizados os formatos de compressão gzip, bzip2, LZMA, XZ ou LZO.

O Initramfs é normalmente alocado em memória pelo *bootloader*. Assim que inicializado o núcleo, este faz a descompressão do ficheiro, caso seja necessário, e utiliza-o como sendo o seu sistema de ficheiros principal, ficando este alocado em memória RAM. Isto leva a que a criação de um Initramfs seja específica para um determinado núcleo de forma a poder incorporar os módulos necessários e específicos do mesmo.

Terminado o processo de montagem do Initramfs, por defeito é executado o ficheiro */init* deste, podendo ser definido outro ficheiro através da inclusão da opção "*rdinit=<ficheiro>*" nos parâmetros de arranque do núcleo, passados normalmente pelo *bootloader*. O processo despoletado no Initramfs normalmente trata de preparar, montar e inicializar o sistema de ficheiros real do sistema operativo. A inicialização deste, por defeito, é feita através do ficheiro */sbin/init*, podendo também este ser definido nos parâmetros de arranque do núcleo utilizando a opção "*init=<ficheiro>*" [5].

2.1.2 Squashfs

O Squashfs é um sistema de ficheiros desenvolvido para sistemas Linux, com apenas permissões de leitura, ou seja, sem permissões de escrita, devido a este se tratar de um sistema de ficheiros comprimido. O Squashfs comprime ficheiros, *inodes* e diretórios, e suporta blocos com tamanho até 1 *MB* para maior compressão. O Squashfs foi desenvolvido para uso geral de sistemas de ficheiros, de apenas leitura, em máquinas com limitações de memória (ex. sistemas embutidos), devido a permitir reduzir o tamanho total deste. A versão original do Squashfs apenas suportava compressão gzip, no entanto, os núcleos Linux mais atuais suportam também compressão LZMA/LZMA2 e LZO [6].

2.2 Métodos de arranque

Quando um sistema computacional é iniciado, este começa sempre por executar o código presente na memória permanente (ROM). Nos computadores pessoais mais comuns, o programa alocado nesta memória é normalmente a BIOS (*Basic Input/Output System*), e mais recentemente UEFI (*Unified Extensible Firmware Interface*). A principal tarefa destes programas é a inicialização global do sistema computacional e *hardware* presente.

Após completa a fase de inicialização, a execução do sistema computacional é passada para a aplicação presente no *Master Boot Record* (MBR) do dispositivo de armazenamento de arranque definido. Normalmente é possível definir previamente qual o dispositivo de armazenamento de arranque, ou seja, para onde deve ser encaminhada a execução do sistema computacional após a fase de inicialização.

O MBR de um dispositivo de armazenamento corresponde aos primeiros 512 *bytes* deste, seguindo-se a informação da tabela de particionamento do dispositivo. Existem várias formas de organizar a informação do MBR mas sempre com 512 *bytes* de tamanho total e a aplicação de arranque nos primeiros *bytes*.

Esta forma de arranque dos sistemas computacionais permite que estes possam ser geridos por diferentes sistemas operativos e que estes possam ser alterados sem comprometer a fase de inicialização do *hardware*, estando assim a execução do sistema computacional apenas dependente do código presente no MBR do dispositivo de arranque, sendo normalmente as aplicações utilizadas no MBR nomeadas de *bootloaders*.

2.2.1 Linux

O arranque dos sistemas Linux, como qualquer outro sistema operativo, é auxiliado por um *bootloader* que se encontra normalmente alocado no MBR do dispositivo de arranque, logo o *bootloader* é o primeiro programa a ser executado após a inicialização da BIOS.

Existem diversos *bootloaders* para sistemas Linux, cada um com as suas características, mas todos eles permitem instalar, em memória RAM, o núcleo do sistema operativo, os parâmetros de arranque e um Initramfs (opcional), passando de seguida o controlo do sistema computacional para o núcleo instalado.

O processo seguinte depende da opção (ou não) pela utilização de um Initramfs. Caso o arranque do sistema operativo seja efetuado sem o auxílio de um Initramfs, é o próprio núcleo que se encarrega de todo o processo de montagem e arranque do sistema operativo, consoante as opções incluídas nos parâmetros do mesmo, sendo no entanto a customização do arranque limitada. Caso contrário, será o Initramfs a realizar todo esse trabalho, como descrito anteriormente (ver secção 2.1.1), removendo assim as limitações impostas pelo núcleo na customização do arranque do sistema operativo (ver figura 2.1) [7].

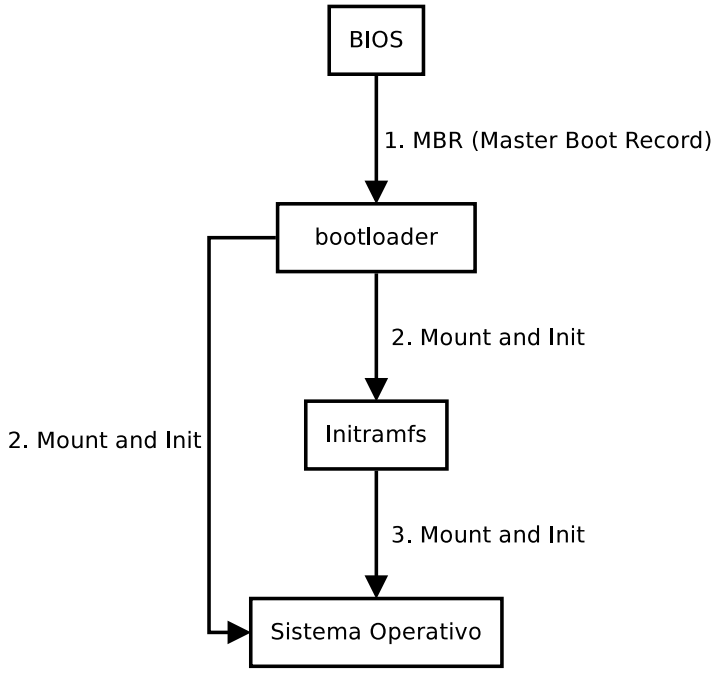


Figura 2.1: Diagrama ilustrativo do arranque de sistemas operativos Linux

Casper/Live

O método de arranque Casper foi concebido de forma a permitir utilizar um sistema operativo sem a necessidade de instalação prévia, ou seja, permite arrancar um sistema operativo Linux pré configurado, armazenado num dispositivo de armazenamento de dados, em qualquer sistema computacional, desde que este possua os requisitos mínimos para o correto funcionamento do mesmo.

Assim, utilizando o método Casper, é possível iniciar o mesmo sistema operativo em diferentes sistemas computacionais, utilizando o mesmo dispositivo de armazenamento de

dados.

Para isso, é necessário que o Initramfs utilizado no arranque do sistema computacional, contenha o método de arranque Casper e que esteja presente nos parâmetros de arranque do núcleo a opção "*boot=casper*". Esta opção permite identificar qual o método de arranque que deve ser utilizado na execução do Initramfs.

Como o método de arranque Casper atua no Initramfs, o restante processo de arranque permanece igual, estando a diferença na forma de montagem do sistema de ficheiros real do sistema operativo. De forma a diminuir o tamanho do sistema de ficheiros real, é utilizada uma imagem deste, presente no dispositivo de armazenamento de arranque, no formato Squashfs (ver secção 2.1.2), e montado um *overlay* em memória RAM sobre este como sendo a raiz do sistema operativo, permitindo assim escrever no sistema de ficheiros do sistema operativo, removendo a limitação de escrita do sistema de ficheiros Squashfs, apesar desta informação não persistir após o encerramento do sistema operativo.

Assim, durante a execução do sistema operativo, as operações de leitura ao sistema de ficheiros são feitas sobre a imagem Squashfs presente no dispositivo de armazenamento e as operações de escrita são feitas na partição criada em memória RAM (*overlay)* [8].

PXE

O PXE (*Preboot eXecution Environment*) é um método de arranque remoto que utiliza uma arquitetura do tipo cliente-servidor, ou seja, o conteúdo do sistema operativo não se encontra em nenhum dispositivo de armazenamento de dados do sistema computacional (cliente) que o vai arrancar, sendo este transferido de outro sistema computacional (servidor) durante o arranque e execução do sistema operativo. Esta tecnologia permite assim executar um sistema operativo num sistema computacional, sem a necessidade de utilização de dispositivos de armazenamento de dados persistentes. Este tipo de utilização é também conhecida como sistemas *diskless*.

Para que seja possível executar este método de arranque, a placa *Ethernet* do cliente é utilizada como dispositivo de arranque, o que implica que exista um *Master Boot Record* (MBR) no *firmware* desta, contendo o código necessário para inicializar o sistema operativo, ou seja, o método de arranque PXE.

Iniciado o método de arranque PXE, o cliente começa por identificar se existem servidores de arranque PXE disponíveis na rede presente. Estes servidores anunciam a sua existência utilizando a opção *DHCPOFFER* do protocolo DHCP, onde é indicado qual o seu IP e quais as opções disponíveis, podendo cada opção representar um sistema operativo diferente, ou até o mesmo sistema operativo mas com opções de arranque diferentes (ex. *debug).*

De seguida, e após ter adquirido um endereço IP utilizando o protocolo DHCP, o cliente obtém junto de um servidor PXE informação relativamente ao núcleo, parâmetros de arranque e Initamfs, de uma determinada opção deste, de modo a possibilitar assim a transferência destes para memória, utilizando o protocolo TFTP (*Trivial File Transfer Protocol*).

Após a conclusão das transferências, é efetuado o processo normal de arranque utilizando os ficheiros transferidos, ou seja, é montado o Initramfs em memória RAM e executado o núcleo, sendo o restante processo de arranque executado consoante os dados obtidos. Normalmente é montada uma partição NFS (Network File System [9] [10]) durante a execução do Initramfs, como sendo a raiz do sistema operativo, de modo a apenas ser transferida a informação do sistema de ficheiros utilizada pelos clientes, ou seja, cada bloco de dados do sistema de ficheiros dos clientes apenas é transferido no momento da primeira operação de

leitura do mesmo.

Os serviços PXE, TFTP e NFS podem ser disponibilizados por um único servidor, ou distribuídos por diferentes servidores independentes (ver figura 2.2) [11] [12].

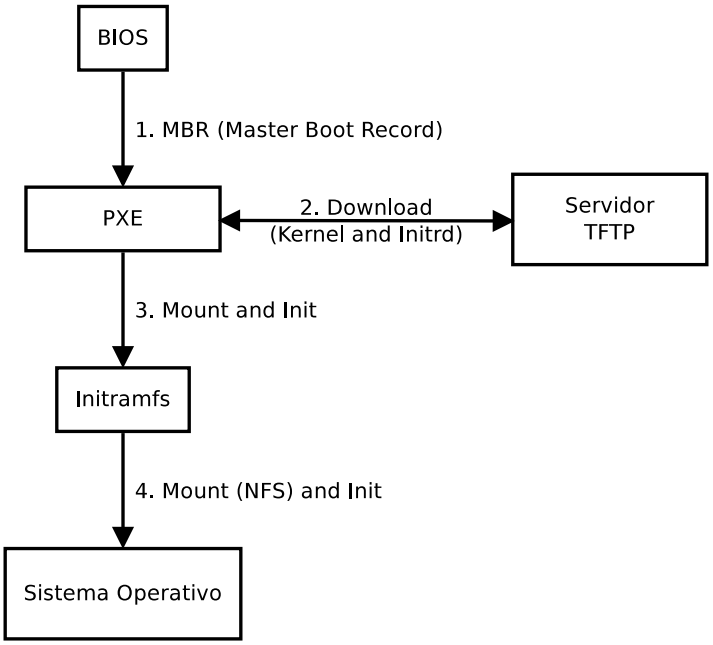


Figura 2.2: Diagrama ilustrativo do arranque de sistemas operativos utilizando a tecnologia PXE

2.3 kexec

O kexec é uma ferramenta disponível para sistemas operativos Linux, que utiliza uma *system call* (*kexec_loader*) do núcleo em execução para carregar e executar um novo núcleo, instalando o novo núcleo na mesma zona de memória onde se encontra o núcleo atual. Para isso, é necessário que o núcleo corrente tenha sido compilado com a opção "*CONFIG_KEXEC=y*" de forma a este estar preparado para a mudança de núcleo.

Para além de alocar em memória um novo núcleo, a *system call* permite também instalar em conjunto com o núcleo, um sistema de ficheiros inicial (Initramfs) e os parâmetros de arranque do núcleo, permitindo assim que estes não se percam durante o *softreset* ao sistema realizado pelo kexec.

As fases de carregamento e execução do kexec podem ser efetuadas em separado, permitindo assim executar outras tarefas entre as duas fases, sendo normalmente a tarefa mais usual o encerramento das aplicações em execução.

Inicializada a execução do kexec, o núcleo corrente é substituído pelo novo núcleo, a memória RAM é libertada, exceto os dados alocados pela *system call* (*kexec_loader*), e executado diretamente o novo núcleo, descartando assim a fase de inicialização da BIOS e do *bootloader*. Devido a isto, o kexec é bastante utilizado para reiniciar sistemas operativos, pois permite acelerar este processo, visto ser poupado o tempo gasto durante a execução da BIOS e do *bootloader*.

Como a fase da BIOS é descartada durante a execução do kexec, o *hardware* do sistema computacional não é reinicializado e o estado do *firmware* dos dispositivos é mantido, existindo assim a possibilidade de ocorrer alguma instabilidade no *hardware* do sistema computacional.

De um modo geral, o kexec pode ser visto como um *bootloader* que pode ser executado em qualquer fase de execução de um sistema operativo, ao invés de restrito à fase inicial de arranque do mesmo [13] [14].

2.4 Redes WiFi

Com o aumento da mobilidade dos equipamentos informáticos, as redes cabladas tornaram-se um entrave a essa mesma mobilidade, fazendo com que as redes sem fios WiFi ganhassem importância nos dias de hoje, sendo já praticamente indispensáveis para qualquer dispositivo móvel (e não só).

Embora as redes sem fios WiFi permitam obter melhor mobilidade, estas têm um alcance limitado, estão sujeitas a maiores interferências externas e as taxas de transferência de dados são em geral mais baixas do que as redes cabladas. No entanto, estas limitações têm vindo a ser corrigidas e melhoradas com o amadurecimento da tecnologia.

De forma a poder restringir o acesso às redes WiFi, estas podem ser protegidas utilizando diferentes métodos de segurança, sendo estes WEP, WPA e WPA2.

Existem dois tipos de arquitetura possível para este tipo de redes: (1) estruturadas, onde a comunicação é feita através de um *Access Point* (AP), ou (2) ad-hoc, onde a comunicação é gerida pelos próprios intervenientes da rede [15].

2.4.1 Estruturadas

As redes sem fios WiFi estruturadas podem ser vistas como um modelo cliente-servidor, em que a comunicação entre dois nós é sempre feita através de um AP (*Access Point*), tanto entre nós dentro da mesma rede, como entre nós de redes diferentes ou o acesso a redes externas (ex. Internet). Não existe comunicação direta entre os nós da rede, sendo esta sempre feita através de um AP, mesmo que os nós estejam ao alcance um do outro.

Neste tipo de redes, para uma estação pertencer à mesma, é necessário que esta se associe a um AP, de modo a este incluir a estação na rede e, caso a rede esteja protegida, é necessário autenticar-se perante o AP, utilizando o método de segurança definido neste.

Desta forma, o AP tem total controlo sobre a rede, sendo a gestão desta da inteira responsabilidade do mesmo (ver figura 2.3).

2.4.2 Ad-hoc

Ao contrário das redes WiFi estruturadas, as rede WiFi ad-hoc podem ser vistas como uma arquitetura P2P (*peer-to-peer*), em que a comunicação é feita diretamente entre os nós da rede, sem a interversão de terceiros, ou seja, sem intermediários na comunicação entre dois nós. Assim para que possa existir comunicação entre dois nós da mesma rede, é necessário que ambos estejam no alcance um do outro.

Deste modo, nas redes ad-hoc, não existe nenhum nó responsável pela gestão da rede, sendo esta feita por todos os seus intervenientes e embora não exista associação por parte dos intervenientes neste tipo de redes, é possível proteger a rede utilizando os métodos de segurança das redes WiFi, referidos anteriormente (ver secção 2.4). Visto não existir nenhuma

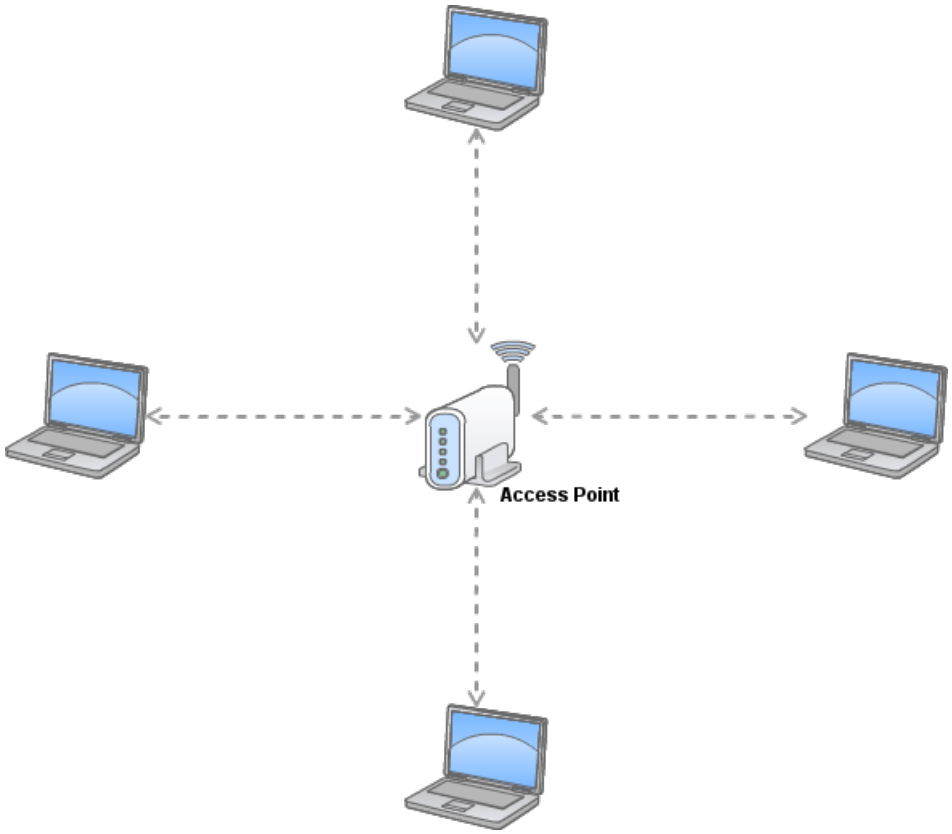


Figura 2.3: Diagrama ilustrativo de uma rede sem fios WiFi estruturada

estação principal responsável pela rede, o acesso a redes externas é da inteira responsabilidade de cada estação, podendo no entanto cada estação partilhar as suas ligações externas com as restantes estações, por exemplo, caso uma estação tenha acesso à Internet, esta pode ser partilhada com as restantes, sendo a comunicação encaminhada pela estação que partilha a mesma (ver figura 2.4).

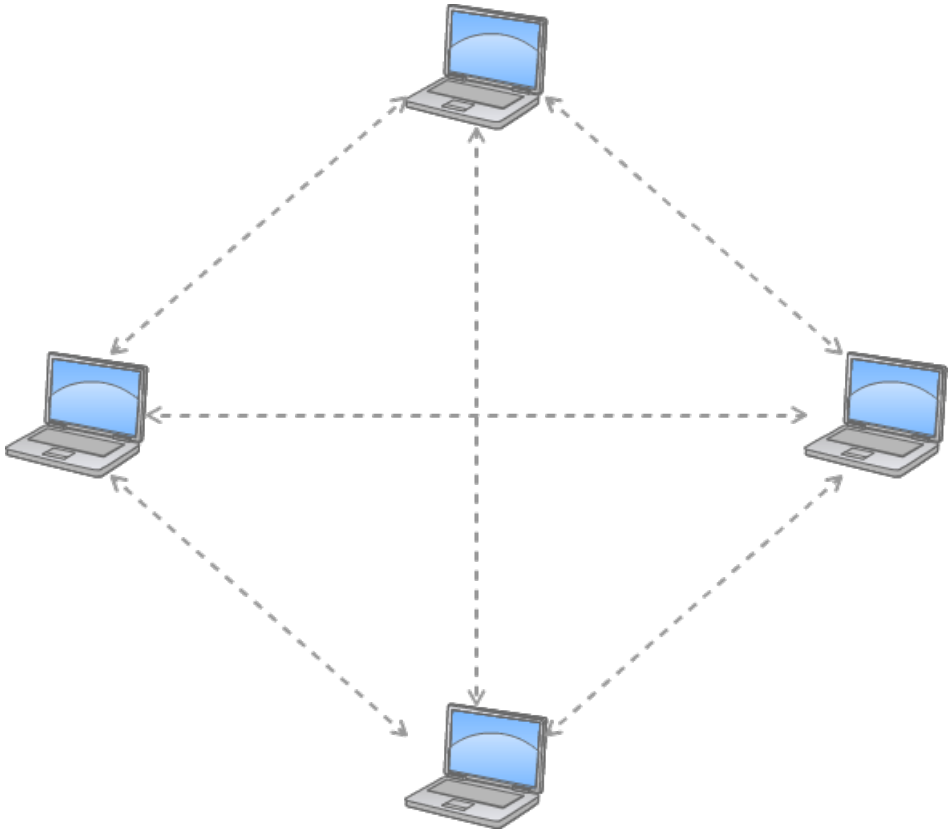


Figura 2.4: Diagrama ilustrativo de uma rede sem fios WiFi ad-hoc

2.5 Códigos Fountain

Os códigos Fountain são uma técnica de *network coding*, ou seja, uma técnica de codificação de dados propícia a comunicações com probabilidade de falhas em que não é possível, ou não é desejável, obter *feedback* por parte dos recetores, permitindo assim transferir informação sem necessidade de notificações sobre o estado de entrega dos pacotes.

Um código Fountain ótimo é aquele que possibilita que um dado conteúdo, dividido em k símbolos, seja possível de recuperar utilizando quaisquer k blocos codificados.

A ideia base subjacente à transmissão de dados com códigos Fountain é existir um codificador que funciona como uma fonte, estando este constantemente a gerar blocos codificados, a partir de k símbolos originais. Quaisquer que sejam os blocos adquiridos por um descodificador, este consegue recuperar os k símbolos originais utilizando k blocos codificados.

No entanto, o ponto ótimo dos códigos Fountain é muito difícil de atingir, sendo necessário sempre um pouco mais do que os k blocos codificados para se conseguir descodificar os k

símbolos originais.

Existem várias implementações diferentes deste tipo de códigos, tendo sido a primeira desenvolvida por Michael Luby e denominada *LT codes* (*Luby Transform code*). As implementações mais recentes destes códigos permitem ter menor complexidade de codificação e descodificação, sendo os *Raptor codes* os mais eficientes hoje em dia, conseguindo tempos de codificação e descodificação lineares [16] [17] [18] [19] [20].

2.5.1 LT Codes

Os *LT Codes* foram a primeira implementação prática, perto do ótimo, dos códigos Fountain, desenvolvido por Michael Luby em 1998 e publicado em 2002. Estes códigos utilizam matrizes esparsas nas tarefas de codificação e descodificação, sendo caracterizados pela utilização da operação lógica XOR (ou exclusivo, $\oplus$) no processo de codificação e descodificação de mensagens.

Para melhor descrever o processo de codificação e descodificação destes códigos, serão utilizadas as seguintes definições:

- a mensagem original é divida em k partes de igual tamanho;
- cada parte da mensagem original é designada por símbolo;
- uma *Codeword* é uma combinação linear de símbolos da mensagem;
- o grau de uma *Codeword* representa o número de símbolos codificados na mesma.

Codificação

No processo de codificação dos *LT Codes*, uma *Codeword* é o resultado da operação lógica XOR de vários símbolos, sendo estes símbolos escolhidos através um algoritmo pseudo-aleatório, desde de que seja utilizado o mesmo algoritmo no processo de descodificação.

$$Codeword = S_1 \oplus S_2 \oplus S_3 \oplus ... \oplus S_n \tag{2.1}$$

A escolha do grau das *Codewords* no momento da codificação é extremamente relevante para a otimização da descodificação dos símbolos. Concretamente, deve-se assegurar que o grau das *Codewords* respeita certas condições, garantindo uma descodificação de todos os símbolos originais perto do ótimo. Para assegurar essas condições, o grau das *Codewords* é determinado segundo uma distribuição de probabilidade discreta específica, descrita de seguida.

Distribuição do Grau

A função de distribuição do grau das *Codewords* inicialmente implementada para os *LT Codes*, foi designada por *ideal soliton*, com a seguinte equação:

$$\begin{aligned} &\rho(1) = 1/K \\ &\rho(d) = \frac{1}{d(d-1)} \quad \text{para d = 2,3,...,K} \end{aligned} \tag{2.2}$$

Como se pode observar na equação, a probabilidade de um dado grau diminui à medida que este aumenta, ou seja, quanto maior o grau menor a sua probabilidade. No entanto, na prática verificou-se que com esta distribuição não se conseguia obter os resultados pretendidos. Isto porque aquando da descodificação, não haveria *Codewords* de grau 1 suficientes para completar a descodificação, alongando o número de *Codewords* necessárias. A solução encontrada foi modificar a distribuição *ideal soliton* de modo a compensar o número de *Codewords* com grau 1, otimizando assim o processo de descodificação. A esta distribuição deu-se o nome de *robust soliton*.

A distribuição *robust soliton*(μ) depende de dois parâmetros, c e δ, que podem ser alterados de modo a controlar o número de *Codewords* com grau 1 geradas na codificação.

Assim, dada a equação seguinte:

$$S = c\log_{\epsilon}(K/\delta\sqrt{K})$$

$$\tau = \begin{cases} \frac{S}{K}\frac{1}{d} & \text{para } d = 1, 2, ..., (K/S) - 1 \\ \frac{S}{K}\log(S/\delta) & \text{para } d = K/S \\ 0 & \text{para } d > K/S \end{cases} \tag{2.3}$$

A *robust soliton* é definida pela seguinte equação:

$$\begin{aligned} Z &= \sum_{d} \rho(d) + \tau(d) \\ \mu(d) &= \frac{\rho(d) + \tau(d)}{Z} \end{aligned} \tag{2.4}$$

Descodificação

O processo de descodificação é feito utilizando *Codewords* de grau 1, ou seja, blocos descodificados que representam um dos k símbolos originais. Assim, e assumindo que o descodificador sabe qual o índice dos símbolos que compõem uma *Codeword*, este procede da seguinte forma:

1. escolhe uma *Codeword* de grau 1 (símbolo) ainda não utilizado;
2. procura quais as *Codewords* de grau superior a 1 dependentes do símbolo escolhido;
3. adiciona o símbolo, utilizando a operação lógica XOR ($\oplus$), às *Codewords* selecionadas no ponto anterior;
4. atualiza o estado das *Codewords*;

Este ciclo é repetido enquanto o número de *Codewords* descodificadas e diferentes (símbolos) for menor que k.

Iteração	**Símbolos**	**Codewords**	**Operação**
1	$S1, S2$	$C1 \mapsto S1 \oplus S2 \oplus S3$ $C2 \mapsto S3 \oplus S4$	$C1 = C1 \oplus S1$ $= S1 \oplus S2 \oplus S3 \oplus S1$ $= S2 \oplus S3$
2	$S1, S2$	$C1 \mapsto S2 \oplus S3$ $C2 \mapsto S3 \oplus S4$	$C1 = C1 \oplus S2$ $= S2 \oplus S3 \oplus S2$ $= S3$
3	$S1, S2, S3$	$C2 \mapsto S3 \oplus S4$	$C2 = C2 \oplus S3$ $= S3 \oplus S4 \oplus S3$ $= S4$
4	$S1, S2, S3, S4$		

Tabela 2.1: Exemplo ilustrativo de uma descodificação com 2 símbolos e 2 *Codewords*

Capítulo 3

Arquitetura do DETIboot

A arquitetura do sistema DETIboot baseia-se num modelo cliente-servidor, onde existe um nó na rede responsável pela transmissão do sistema operativo (servidor), e os restantes nós da rede funcionam como clientes, recebendo o sistema operativo transmitido pelo servidor e executando o mesmo após a transferência estar concluída.

Para comodidade dos utilizadores e melhor usabilidade do sistema DETIboot, a comunicação é feita utilizando redes sem fios WiFi, permitindo assim o fácil acesso à transmissão do servidor por parte dos clientes e mobilidade do servidor. Assim, é necessário que os sistemas computacionais, tanto dos clientes como do servidor, estejam munidos de dispositivos de rede sem fios WiFi, internos ou externos, de forma a poderem comunicar entre si.

A rede sem fios WiFi utilizada no sistema DETIboot é do tipo ad-hoc (ver secção 2.4.2), permitindo assim que o servidor não necessite de ter características de *Access Point* (AP) e libertando também o servidor das tarefas de gestão da rede desempenhadas pelos APs, limitando-se a transmitir a informação sem outras preocupações. Também por ser uma rede ad-hoc, os clientes podem entrar e sair da rede, em qualquer instante, sem perturbar a transmissão do servidor, pois não existe a necessidade de associação por parte dos clientes (ver figura 3.1).

3.1 Estudos prévios

Até ser encontrada a solução final do sistema DETIboot, para distribuir e arrancar um sistema operativo em máquinas de terceiros, este sofreu várias transformações de forma a corrigir as limitações que foram surgindo, tanto a nível de segurança como a nível de usabilidade e desempenho.

A solução inicialmente planeada para o sistema DETIboot passava pela utilização de dispositivos de armazenamento de dados portáteis (pen USB) como meio de distribuição do sistema operativo, ou seja, o sistema operativo seria armazenado e arrancado através de um dispositivo de armazenamento USB, sendo o DETIboot uma extensão ao método de arranque Casper (ver secção 2.2.1). Rapidamente se concluiu que este método poderia trazer sérios problemas de segurança, visto que facilmente se poderia adulterar o conteúdo do dispositivo de armazenamento utilizado. Além disso, resultaria numa fraca usabilidade relativamente à distribuição e manipulação do sistema operativo, visto que seria necessário replicar os dados do sistema operativo pelos vários dispositivos de armazenamento de dados sempre que este sofresse alguma alteração, por mais pequena que fosse.

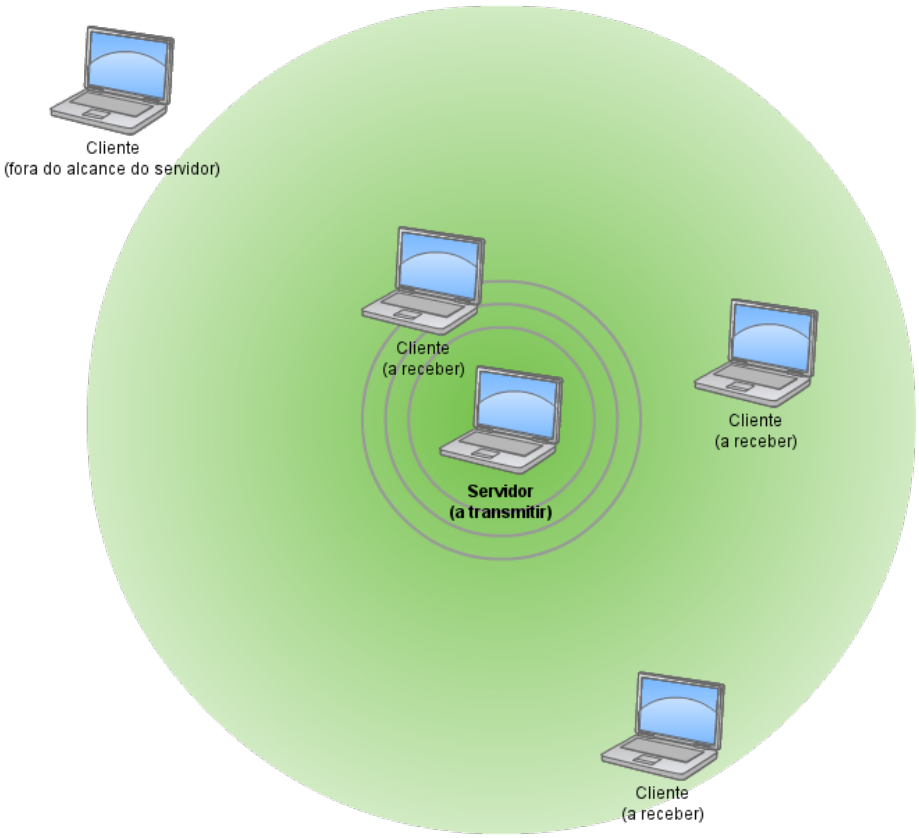


Figura 3.1: Diagrama exemplificativo de uma possível distribuição dos clientes e servidor do sistema DETIboot.

De forma a melhorar a segurança e usabilidade do sistema, abordou-se o problema utilizando a tecnologia PXE (ver secção 2.2.1), permitindo assim ter o conteúdo do sistema operativo (sistema de ficheiros) remoto, com auxílio da tecnologia NFS (*Network File System* [9]), facilitando assim a distribuição do mesmo e permitindo proteger este contra escrita por parte de terceiros. No entanto, e apesar deste método resolver os problemas de segurança e usabilidade da primeira abordagem, o desempenho do sistema ficaria demasiado dependente do número de utilizadores, ou seja, o tempo de arranque do sistema operativo aumentaria consideravelmente com o aumento do número de utilizadores (não escalável).

Assim, com as limitações encontradas nas 2 abordagens anteriores, idealizaram-se os princípios básicos necessários para o correto funcionamento do método de arranque DETIboot:

- Distribuição do sistema de ficheiros remota (melhor usabilidade);
- Utilizadores sem permissões de escrita (melhor segurança);
- Independência relativamente ao número de utilizadores (melhor desempenho).

A partir destes princípios nasceu a solução final para o método de arranque DETIboot, que consiste na utilização de um modelo cliente-servidor numa rede sem fios WiFi (ad-hoc), de modo a permitir uma melhor usabilidade tanto por parte dos clientes como do servidor. O sistema computacional utilizado como servidor, fica assim responsável por transmitir a informação necessária para a execução de um sistema operativo Linux (núcleo e sistema de ficheiros). Para evitar problemas de escalabilidade (melhor desempenho), a transmissão é feita em difusão na rede e são utilizados códigos Fountain para suprimir a perda de pacotes resultante da falta de *feedback* na comunicação por difusão. Este método de transmissão permite que qualquer sistema computacional (cliente) que esteja ao alcance do servidor, receba uma imagem do sistema operativo contento o núcleo e sistema de ficheiros respetivo, em qualquer instante e de forma independente relativamente aos restantes clientes.

3.2 Rede de comunicação

Para que o sistema operativo possa ser acedido remotamente pelos clientes, é necessário que exista comunicação entre estes e o servidor. As redes sem fios WiFi permitem que tal seja possível num determinado espaço geográfico de forma rápida e sem a necessidade de pré-instalação, ao contrário das redes cabladas onde é necessário que o espaço em questão esteja devidamente equipado (ligações por cabo).

No entanto, como descrito na secção 2.4, as redes sem fios WiFi podem ser estruturadas ou ad-hoc. No sistema DETIboot pretendeu-se minimizar os requisitos e responsabilidades do servidor na rede, de forma a conseguir obter a melhor taxa de transmissão de dados possível. Visto que a utilização de redes sem fios WiFi estruturadas no sistema DETIboot implicaria que o servidor tivesse caraterísticas de *Access Point* (AP) e ficasse responsável pela gestão total da rede, incluindo o encaminhamento de todo o tráfego da rede, optou-se pela utilização de uma rede sem fios WiFi do tipo ad-hoc, libertando assim o servidor de tais tarefas e tornando-o responsável por apenas uma única tarefa: difundir o sistema operativo pelos clientes presentes na rede. A opção pelo tipo de rede sem fios WiFi ad-hoc permite também que os clientes possam entrar e sair da rede, em qualquer instante, sem perturbar a transmissão do servidor visto não existir a necessidade de associação por parte dos clientes.

Um aspeto importante que foi tido em conta no desenho da arquitetura do sistema DETIboot é de que este seja escalável, ou seja, suporte um número ilimitado de utilizadores com a menor variação de desempenho possível ou até mesmo ter um desempenho constante, independentemente do número de utilizadores. Assim, optou-se por utilizar comunicação em difusão por parte do servidor de forma a que este tenha uma taxa de transmissão de dados constante e independente do número de clientes presentes na rede. No entanto, as redes sem fios WiFi são bastante propensas a ruído externo, existindo perdas de pacotes diferentes entre os clientes. Devido à inexistência de *feedback* nas comunicações por difusão, também não existe retransmissão dos pacotes perdidos, sendo assim necessária uma solução para suprimir a perda de pacotes resultante do modelo de comunicação adotado.

A solução encontrada para suprimir esta perda foi a utilização de códigos Fountain (ver secção 2.5), criando aleatoriedade nos blocos transmitidos pelo servidor e permitindo que os clientes consigam reconstruir o ficheiro transmitido pelo servidor a partir de quaisquer blocos transmitidos, com o agravamento de serem necessários cerca de 5% mais blocos do que os que compõem o ficheiro original transmitido.

Como a transmissão é feita diretamente e em difusão, não existe necessidade de reencaminhamento de tráfego por parte dos clientes, ou seja, utilização de comunicação IP. Posto isto, não existe atribuição de IPs na rede, logo não é necessário a existência de um serviço DHCP no servidor e a comunicação é feita de forma canónica sobre a camada de ligação de dados (L2), sem a utilização de nenhum protocolo de transporte (L4), nem nenhum protocolo de rede (L3), sendo a camada de ligação de dados responsável por garantir a integridade dos pacotes e a utilização de códigos Fountain para suprimir a perda de pacotes como já referido [21].

No sistema DETIboot, devido à inexistência de *feedback*, apenas o servidor envia pacotes para a rede, limitando-se os clientes a receber os pacotes transmitidos. Para que os clientes possam entender a transmissão do servidor, foi desenhado o protocolo de comunicação *File Broadcast Protocol* (FBP, ver secção 3.2.1), permitindo ao servidor transmitir uma *Codeword* por pacote. Em cada trama é também incluída informação relevante sobre a *Codeword* e a mensagem original, como o número de símbolos que compõem a mensagem, a semente utilizada na geração dos índices da *Codeword* e o grau da *Codeword*. Esta informação é vital para que os clientes consigam descodificar as *Codewords* e reconstruir o ficheiro transmitido pelo servidor (ver figura 3.2).

3.2.1 Protocolo FBP

O protocolo desenhado para a comunicação entre o servidor e os clientes foi feito de modo a permitir a transmissão de blocos de dados codificados (*Codewords*) por parte do servidor, e descodificação dos mesmos por parte dos clientes, tendo como principal objetivo maximizar o tamanho do bloco de dados de cada trama, diminuindo assim o número total de tramas necessárias por ficheiro. Assim, cada trama é composta pelos seguintes campos:

- `k`: número total de símbolos que compõem o ficheiro original;
- `seed`: semente utilizada na geração da *Codeword*;
- `degree`: grau da *Codeword*;
- `data`: bloco de dados codificados (*Codeword*);

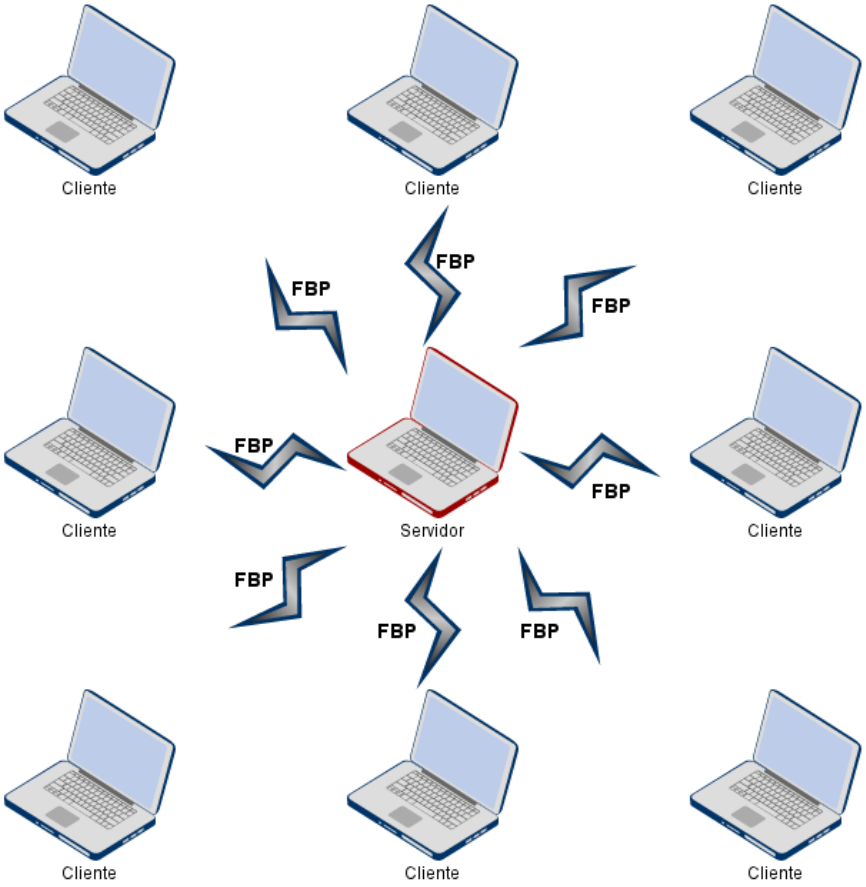


Figura 3.2: Diagrama ilustrativo da comunicação WiFi, utilizando pacotes FBP.

k	seed	degree	data
4 octetos	4 octetos	4 octetos	0-1488 octetos

Tabela 3.1: File Broadcast Protocol (FBP) - Protocolo de rede (L3), identificado através do código/tipo 0x1986.

Para a descodificação de uma *Codeword* é necessário conhecer os índices dos blocos originais que compõem a mesma. A especificação direta destes no protocolo implicaria a diminuição do tamanho do bloco de dados codificados de cada trama ou até a utilização de outra trama suplementar com essa indicação, prejudicando assim o desempenho da transmissão.

A solução encontrada para este problema foi incluir em cada trama a semente utilizada pelo codificador na geração do bloco de dados codificado, ou seja, a semente utilizada para gerar pseudo-aleatoriamente os índices dos blocos combinados. Para além da semente (`seed`) e do bloco de dados codificado (`data`), é necessário ainda que o servidor transmita o número total de símbolos que compõem o ficheiro (`k`) e o número de blocos combinados na *Codeword* (`degree`) e que ambos (cliente e servidor) usem a mesma função pseudo-aleatória.

Assim, utilizando este protocolo, o descodificador (cliente) ao receber uma trama consegue extrair desta o número total de símbolos que compõem o ficheiro (`k`), o bloco de dados codificados (`data`), a semente utilizada pelo codificador na geração da *Codeword* (`seed`) e o grau da *Codeword* (`degree`). Através da semente e do grau, o descodificador consegue replicar os índices gerados pelo codificador, ou seja, os índices da *Codeword*, visto se tratar de uma geração pseudo-aleatória.

Durante a geração dos índices de uma *Codeword*, é necessário garantir que estes não se repetem de modo a evitar ter blocos repetidos na mesma *Codeword*. Para isso, é utilizado um *array* de dimensão igual a `k` para auxiliar a geração de índices de forma a garantir que estes não se repetem, segundo o seguinte algoritmo:

```
empty_val=0xFFFFFFFF; // definição do valor de vazio

// função de inicialização do array auxiliar com o valor vazio
initRand(int k) {
  for(i=0 ; i<k ; i++) {
    array[i]=empty_val;
  }
  set_rand(seed); // semente a utilizar na geração pseudo-aleatória
  r = k; // limite de utilização do array (máximo)
}

// função de geração de um número aleatório sem repetição
getRand() {
  i = rand() % r; // i = valor pseudo-aleatório entre 0 e r
  result = array[i] == empty_val ? i : array[i];
  array[i] = array[--r] == empty_val ? r : array[r];
  return result;
```

```
}

initRand(k);
for(i=0 ; i < degree ; i++) {
  index[i] = getRand();
}
```

Este algoritmo trata de inicializar um *array* auxiliar com o valor definido como vazio, de modo a indicar que todas as posições do *array* se encontram livres. É também feita na fase de inicialização a definição da semente (`seed`) a utilizar na geração pseudo-aleatória e a definição do limite de utilização do *array* (`r`), sendo este igual a `k`.

Após a fase de inicialização, sempre que solicitado um novo número, é gerado pseudo-aleatoriamente um número entre 0 e `r` (`i`). Caso a posição do *array* respetiva ao número gerado esteja livre (`array[i]==empty_val`) é devolvido o número gerado (`i`), caso contrário é devolvido o valor presente na posição do *array* respetiva (`array[i]`).

Antes da devolução do valor gerado, é decrementado o valor de `r` e atualizado o valor da posição do *array* utilizado (`array[i]`) com um valor ainda não gerado, sendo este `r` caso o valor correspondente no *array* seja vazio (`array[r]==empty_val`), caso contrário é atualizado com o valor ainda não gerado presente na posição do *array* respetiva(`array[r]`).

Desta forma, garante-se que é sempre gerado um novo número entre 0 e `k`, sem colidir com os números gerados anteriormente.

3.3 Cliente

Na arquitetura do sistema DETIboot, os clientes são máquinas que recebem e arrancam o sistema operativo transmitido pelo servidor. Para um determinado sistema computacional funcionar como cliente do sistema DETIboot, é necessário que este execute o método de arranque DETIboot na sua fase de arranque, ou seja, antes de iniciado o sistema operativo.

Assim, qualquer máquina que cumpra os requisitos mínimos (ver secção 3.3.1) necessários pode receber (ver secção 3.3.2) e armazenar (ver secção 3.3.3) um sistema operativo transmitido pelo servidor, e de seguida arrancar (ver secção 3.3.4) o mesmo.

Para que todo o processo de receção, armazenamento e arranque do sistema operativo seja feito na fase de arranque da máquina cliente, optou-se pela utilização de um núcleo Linux e Initramfs respetivo, contendo os programas de arranque do DETIboot e os *drivers wireless* necessários para a utilização de dispositivos de rede sem fios WiFi.

No entanto, os programas de arranque do DETIboot têm de lidar com máquinas muito variadas, o que poderá criar algumas limitações e variações de desempenho (ver capítulo 5).

3.3.1 Requisitos mínimos

Para a execução do método de arranque DETIboot, o cliente necessita de possuir um dispositivo de comunicação sem fios WiFi, sendo que todos os computadores portáteis hoje em dia já vêm com este tipo de dispositivos integrados. Para além disso, é ainda essencial que a máquina cliente tenha acesso aos programas necessários para a execução do método de arranque DETIboot, sendo o mais natural estes estarem alojados num dispositivo de

armazenamento de dados, interno ou externo, acessível na fase de arranque das máquinas em questão.

3.3.2 Receção do SO

No método de arranque DETIboot, os clientes recebem o sistema operativo através da transferência de uma imagem do mesmo, emitida por um determinado servidor numa rede sem fios WiFi ad-hoc. Esta transferência é feita com códigos Fountain (ver secção 2.5) onde os blocos da imagem transmitida são codificados de forma a permitir aos clientes reconstruirem a mesma a partir de quaisquer blocos adquiridos. Esta forma de transferência permite suprimir a perda de pacotes dos clientes sem a necessidade de *feedback*, sendo no entanto necessário adquirir cerca de 5% mais blocos do que os que compõem a imagem emitida.

3.3.3 Armazenamento do SO

Após a receção do sistema operativo, é necessário armazenar o seu sistema de ficheiros na máquina cliente de modo a permitir o posterior arranque do mesmo. Como o DETIboot foi desenhado para a realização de exames práticos, é necessário garantir que tudo se passa sem deixar "marca" nos clientes (não fazem estragos), portanto a opção pelo armazenamento em memória RAM é a mais indicada. No entanto, esta acarreta alguns problemas, tais como a limitação deste tipo de armazenamento nos computadores portáteis, sendo necessário garantir que os clientes possuam memória RAM suficiente para armazenar o sistema operativo. Outro inconveniente do armazenamento do sistema operativo em memória RAM é que durante a mudança de núcleo (kexec, ver secção 2.3) esta memória é libertada, sendo apenas mantidos os dados referentes ao núcleo e Initramfs carregado. A resolução deste problema é feita na fase de construção da imagem do sistema operativo, tendo sido encontradas duas soluções:

- Utilizar distribuições cujo sistema de ficheiros seja no formato Initramfs (ex. Slitaz);
- Incluir o sistema de ficheiros no Initramfs de arranque.

Apesar desta abordagem pelo armazenamento do sistema de ficheiro em memória RAM aumentar o consumo desta, as operações de leitura e escrita tornam-se mais rápidas devido à velocidade de acesso das mesmas ser maior em memória RAM do que em dispositivos de armazenamento de dados persistentes, e permite que o sistema operativo seja automaticamente descartado após a realização da prova devido à memória RAM ser volátil. Isto permite também que embora o sistema operativo possa ser obtido e analisado pelos alunos de forma a tentarem contornar as restrições impostas pelos docentes durante a sua configuração, estes não o conseguirão em tempo útil.

3.3.4 Arranque do SO

O arranque do sistema operativo é auxiliado pela ferramenta kexec (ver secção 2.3), permitindo utilizar o núcleo e arrancar o sistema operativo importados sem a necessidade de os transferir permanentemente para um dispositivo de armazenamento de dados e sem a necessidade de reiniciar completamente o sistema computacional. O restante processo de arranque, após a execução do kexec, é da inteira responsabilidade do sistema operativo importado. De modo a permitir alguma customização do arranque do sistema operativo, inclusive até transferir este para um dispositivo de armazenamento de dados persistente e arrancar a partir do

mesmo, as instrução de arranque do sistema operativo devem ser passadas através da inclusão do ficheiro "startup" contendo as mesmas, na raiz da imagem transmitida (ver figura 3.3).

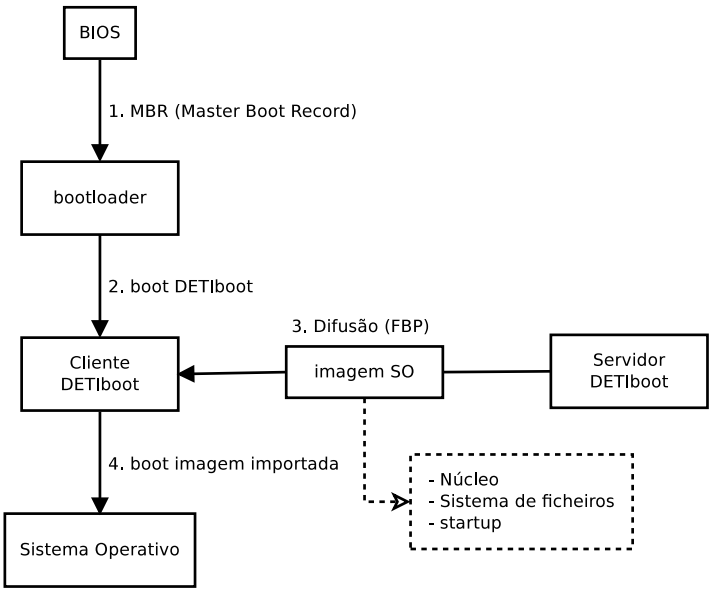


Figura 3.3: Diagrama ilustrativo do arranque de sistemas operativos utilizando o método de arranque DETIboot

3.4 Servidor

Na arquitetura do sistema DETIboot, o servidor é a estação responsável pela transmissão do sistema operativo para todos os clientes alcançáveis pelo mesmo e presentes na rede. Ao contrário das máquinas dos clientes, para um determinado sistema computacional funcionar como servidor do sistema DETIboot, é necessário que este execute o processo de transmissão da imagem de um sistema operativo após a inicialização do seu próprio sistema operativo.

No sistema DETIboot, pode ser utilizado qualquer tipo de sistema computacional como servidor, desde máquinas com grande capacidade de processamento, *routers* ou até os novos mini-PCs, desde que possuam os requisitos mínimos (ver secção 3.4.1) necessários para armazenar (ver secção 3.4.2) e emitir (ver secção 3.4.3) um sistema operativo. Porém, as características deste e a dimensão do sistema operativo difundido para os clientes pode influenciar o desempenho global do sistema (ver capítulo 5).

3.4.1 Requisitos mínimos

Para a transmissão de um sistema operativo, o servidor necessita possuir um dispositivo de comunicação sem fios WiFi, interno ou externo, de forma a possibilitar a comunicação com os clientes. É ainda necessário que o servidor disponha de acesso à imagem do sistema operativo a transmitir.

Outros dois requisitos, embora não sejam essenciais mas que podem influenciar drasticamente o desempenho do servidor são: a capacidade de memória RAM do mesmo, sendo necessário que o servidor tenha memória RAM livre suficiente para armazenar a imagem do

sistema operativo a transmitir, evitando a utilização de memória virtual que pode prejudicar bastante a taxa de transmissão de dados do servidor devido a criar uma atividade de paginação muito intensa. Por defeito, a taxa de transmissão de pacotes por difusão nas redes WiFi é limitada a 1 *Mb/s*, sendo necessário que o servidor possua mecanismos para definir o *basic rate* e *multicast rate* da interface de rede WiFi utilizada, de modo a aumentar a taxa de transmissão do sistema operativo.

3.4.2 Armazenamento do SO

Para emitir um sistema operativo, o servidor necessita de ter acesso ao conteúdo do mesmo, ou seja, necessita de ter acesso a uma imagem do sistema operativo. Esta imagem pode ser armazenada em qualquer dispositivo de armazenamento de dados, interno ou externo, podendo no entanto a velocidade de acesso a este influenciar o desempenho do servidor, devido a ser necessário efetuar operações de leitura sobre os blocos da mensagem original, sempre que gerada uma nova *Codeword*. A combinação de blocos aleatórios em *Codewords* obriga a aceder aleatoriamente a regiões muito distintas da imagem, o que pode originar um problema de paginação excessiva caso a imagem não consiga caber totalmente em memória RAM durante todo o processo de codificação (ver capítulo 5).

3.4.3 Emissão do SO

A emissão da imagem do sistema operativo por parte do servidor é feita utilizando códigos Fountain (ver secção 2.5) através de uma interface de rede WiFi, previamente configurada numa rede ad-hoc. Esta emissão consiste na geração e difusão constante de *Codewords*, através de pacotes do tipo FBP (ver secção 3.2.1). Assim, o servidor está constantemente a debitar pacotes FBP para a rede, funcionando como uma fonte de dados (*Codewords*), utilizada pelos clientes para reconstruir a imagem do sistema operativo, a partir de quaisquer *Codewords* em qualquer instante.

Devido à comunicação ser feita em difusão em redes WiFi, é necessário definir o *basic rate* e *multicast rate* da interface de rede utilizada pelo servidor de modo a aumentar a taxa de transmissão, visto que, por defeito, a difusão de pacotes em redes WiFi é limitada a uma taxa de 1 *Mb/s* de modo a manter a compatibilidade com todas as normas WiFi (principalmente a 802.11b). Isto implica que a definição do *basic rate* e *multicast rate* pode causar a perda de compatibilidade com normas que não suportem os valores definidos, sendo 54 *Mb/s* o valor máximo mais adequado de forma a manter a compatibilidade com a norma mais utilizada (802.11g), e afetando apenas a norma obsoleta (na prática) 802.11b.

Capítulo 4

Implementação do DETIboot

Neste capítulo é exposta a implementação da solução final do DETIboot bem como toda a investigação feita até se chegar a esta.

Assim, na secção 4.1 é descrita a rede utilizada na comunicação do sistema DETIboot, onde são detalhados os pormenores da mesma. Na secção 4.2 é abordada a implementação dos códigos Fountain, onde é descrito o processo de codificação e descodificação implementado utilizando os mesmos. As aplicações e *scripts* desenvolvidos, necessários para o correto funcionamento do método de arranque DETIboot, são descritos nas secções 4.3 e 4.4 respetivamente. Por último, na secção 4.5 é exposta a integração final de todos os módulos desenvolvidos.

4.1 Rede

Como descrito anteriormente no capítulo 3, a comunicação entre o servidor e os clientes é feita através de uma rede sem fios WiFi, em modo ad-hoc. De forma a auxiliar e minimizar possíveis erros na configuração da rede, foram desenvolvidos 2 *scripts* de configuração dos dispositivos WiFi, descritos mais à frente na secção 4.4, onde o único parâmetro variável é o canal de comunicação, permitindo assim utilizar um canal pouco congestionado ou até ter 2 ou mais servidores a difundir sistemas operativos distintos na mesma área mas em canais diferentes. Os parâmetros fixos são o SSID e o BSSID, garantindo assim que todas as estações permanecem ligadas à mesma rede, mesmo que fora do alcance umas das outras, ou sem qualquer comunicação/transmissão de dados na mesma. Esta rede é criada pelo primeiro sistema computacional a configurar o seu dispositivo WiFi, seja cliente ou servidor. A configuração do cliente e servidor difere apenas na configuração da taxa de transmissão de dados em difusão. A transmissão de pacotes *broadcast* em redes sem fios WiFi é limitada a 1 Mbit/s, sendo assim necessário definir o *basic rate* e *multicast rate* no servidor, de modo a aumentar a taxa de transmissão de pacotes *broadcast* deste. A ferramenta *iw*, disponível em sistemas Linux, permite efetuar tais configurações, sendo assim utilizada pelo servidor para configurar a sua interface de rede. Como as estações cliente não transmitem qualquer informação, não é necessário definir o *basic rate* e *multicast rate* destes, sendo portanto utilizada a ferramenta *iwconfig*, que embora mais limitada suporta interfaces de rede mais antigas, para a configuração das suas interfaces de rede.

Os dados transmitidos pelo servidor realizam a codificação de uma imagem ISO utilizando um algoritmo desenvolvido baseado nos códigos Fountain (ver secção 2.5). A transmissão em

difusão permite distribuir a imagem por todas as estações no seu alcance sem necessidade de *feedback*. Embora possa existir perda de pacotes, este é irrelevante para o processo de descodificação do ficheiro por parte dos clientes, devido à aleatoriedade introduzida pelos códigos Fountain.

O conteúdo desta imagem ISO é composta pelo núcleo, o sistema de ficheiros do sistema operativo no formato Initramfs e o ficheiro "startup" contendo as instruções de arranque do sistema operativo transmitido.

4.2 Códigos Fountain

A codificação e descodificação dos dados transmitidos foi implementada utilizando um algoritmo baseado nos códigos Fountain, mais precisamente nos *LT Codes* (ver secção 2.5.1). Esta implementação foi desenvolvida utilizando a linguagem de programação C++, de forma a tirar partido da programação orientada a objetos, tendo sido criadas as seguintes classes:

- `Codeword`: representa um bloco codificado (ver secção 4.2.1);
- `Symbol`: representa um bloco descodificado (ver secção 4.2.1);
- `RandPerm`: permite gerar um número arbitrário de índices aleatórios sem repetição (ver secção 4.2.1);
- `Encoder`: responsável pela geração de *Codewords* (ver secção 4.2.2);
- `Decoder`: responsável pela descodificação de *Codewords* (ver secção 4.2.3).

Para além destas classes, foram ainda desenvolvidas algumas funções auxiliares em C que permitem, entre outras, gerar o grau das *Codewords* segundo a distribuição *robust soliton* (ver secção 2.5.1), e realizar as operações XOR ($\oplus$) utilizando instruções vectorizadas, caso o processador do sistema computacional as suporte, de modo a melhorar o desempenho dos processos de codificação e descodificação.

Este código foi desenvolvido de forma a possibilitar a codificação e descodificação de dados utilizando códigos Fountain, através das classes Encoder e Decoder respetivamente, em qualquer aplicação desenvolvida em C++, como é o caso das aplicações desenvolvidas no contexto desta dissertação, fbp-server (ver secção 4.3.3) e fbp-cliente (ver secção 4.3.2).

4.2.1 Classes auxiliares

Codeword

Esta classe é utilizada para guardar os blocos de dados codificados, um bloco por objeto, e os seguintes atributos necessários para a sua descodificação:

- `char* data`: referência para o bloco de dados;
- `off_t data_size`: tamanho do bloco de dados;
- `int degree`: grau corrente da *Codeword*;
- `int index`: combinação linear dos índices dos símbolos originais que compõe a *Codeword*;

- `int seed`: semente utilizada na geração da *Codeword*.

Assim sempre que um objeto desta classe é instanciado, é armazenado o bloco de dados codificado em memória, definidos os atributos *data_size*, *degree* e *seed* consoante a *Codeword* armazenada e o atributo *index* é definido através da combinação linear de todos os índices dos blocos que compõe a *Codeword*.

Sempre que é removido um símbolo da *Codeword*, utilizando a operação XOR com um bloco de dados descodificado, o *degree* é decrementado uma unidade e o *index* é o resultado da operação lógica XOR entre o mesmo e o índice do bloco descodificado utilizado.

Symbol

A classe *Symbol* é utilizada para identificar os blocos descodificados e os blocos codificados (*Codewords*) dependentes de um determinado símbolo. Assim, esta classe é composta por apenas 2 atributos:

- `Codeword* decoded_word`: referência para uma *Codeword* descodificada;
- `list<Codeword*> codewords`: lista de referências de *Codewords*.

Durante o processo de descodificação, os objetos cujo símbolo correspondente esteja descodificado, utilizam o atributo *decoded_word* para referenciar a *Codeword* descodificada correspondente. Os símbolos ainda não descodificados utilizam a lista *codewords* para referenciar as *Codewords* dependentes do mesmo.

RandPerm

Na geração de uma *Codeword* são escolhidos x blocos aleatórios da mensagem original e combinados através da operação lógica XOR. Sendo x o grau da *Codeword*, é necessário gerar x índices entre 0 e k, para cada *Codeword*. Como este processo de geração de índices é feito sempre que gerada uma nova *Codeword*, o tempo despendido pelo servidor neste processo pode comprometer o desempenho deste, mais concretamente diminuir a taxa de transmissão, sendo necessário otimizar o mesmo. A classe RandPerm foi desenvolvida para este mesmo propósito permitindo gerar números inteiros aleatórios, sem repetição, entre 0 e um determinado número, de forma otimizada. Assim, a classe RandPerm é composta por 3 atributos:

- `int n`: limite superior dos números gerados;
- `int r`: número de possibilidades restantes;
- `int array`: memória para garantir que os números gerados não se repetem.

No momento de instanciação de um objeto desta classe, é indicado o limite superior dos números a serem gerados (n), sendo a variável r inicializada com este mesmo valor. O *array* é alocado também na fase de inicialização com n números inteiros e definido o valor $0xFFFFFFFF$ em todos eles que indica que um determinado número ainda não foi gerado, ou seja, todos eles.

Sempre que invocada a função de atribuição de um novo número aleatório (*getPerm()*) sobre o objeto, é decrementado o valor de r uma unidade, gerado um número pseudo-aleatório

i, entre 0 e r, e caso $array[i] == 0xFFFFFFFF$ o valor de retorno da função é i, caso contrário o valor de retorno é $array[i]$. Após encontrado o valor de retorno, é necessário atualizar o valor de $array[i]$, sendo este r caso $array[r] == 0xFFFFFFFF$, caso contrário $array[i] = array[r]$. Este processo garante assim que os números gerados não se repetem, de forma eficiente (ver figura 4.1).

4.2.2 Classe Encoder

Esta classe é responsável pela geração das *Codewords*, sendo estas geradas consoante os parâmetros passados a quando a sua instanciação (construtor), permitindo assim definir os dados originais, tamanho das *Codewords* e opcionalmente os parâmetros da distribuição do grau *robust soliton* (c e δ). Após a instanciação da classe, sempre que invocada a função *genCodeword* sobre o objeto instanciado, é gerada uma nova *Codeword*, tendo como valores de retorno a semente utilizada na geração dos índices, o grau da *Codeword* (gerado utilizando a distribuição *robust soliton*) e o bloco de dados codificados, necessários para a descodificação da mesma. Assim, utilizando estes valores retornados em conjunto com o número total de símbolos, facilmente se compõem uma trama FBP (ver secção 3.2.1) para poder ser difundida pelos clientes.

Os parâmetros do construtor desta classe são um ponteiro para os dados a codificar, o tamanho total destes dados (k símbolos), o tamanho de cada *Codeword* e opcionalmente os valores de c e δ referentes à função de distribução do grau, permitindo assim gerar *Codewords* de qualquer tamanho, a partir de qualquer conjunto de dados e manipular a função de distribuição do grau das *Codewords*.

4.2.3 Classe Decoder

Esta classe é responsável pela descodificação de *Codewords*, sendo este processo realizado por uma *thread* específica, de forma a permitir que a aplicação/*thread* principal que utiliza esta classe possa desempenhar outras tarefas, enquanto o processo de descodificação é executado.

Para instanciar esta classe (construtor), é necessário indicar qual o número de símbolos que compõem a informação original (k) e o tamanho de cada *Codeword*.

Após a instanciação da classe, é possível iniciar ou suspender o processo de descodificação, desempenhado por uma *thread* específica, através das funções *start* e *stop*, respetivamente.

Para adicionar *Codewords* ao processo de descodificação, é utilizada a função *addCodeword* que instância um objeto do tipo Codeword com os parâmetros indicados na mesma e guarda uma referência para o objeto criado numa lista, de forma a disponibilizar as *Codewords* à *thread* responsável pela descodificação sem bloquear a *thread* principal, libertando assim esta última para outras tarefas.

Concluído o processo de descodificação, a informação resultante pode ser escrita numa determinada zona de memória ou ficheiro, utilizando a função *writeData*.

O processo de descodificação pode requerer bastante esforço computacional, tanto em termos de processamento como a nível de memória RAM, aumentando consoante o tamanho da mensagem transmitida. Assim de forma a não existir duplicação de *Codewords*, estas são armazenadas uma única vez em memória e utilizadas referências para as mesmas, através de uma lista de k símbolos, contendo cada símbolo referências para as *Codewords* dependentes, permitindo assim uma pesquisa rápida e eficiente pelas *Codewords* dependentes de um determinado símbolo e minimizando o consumo de memória. No caso das *Codewords* já

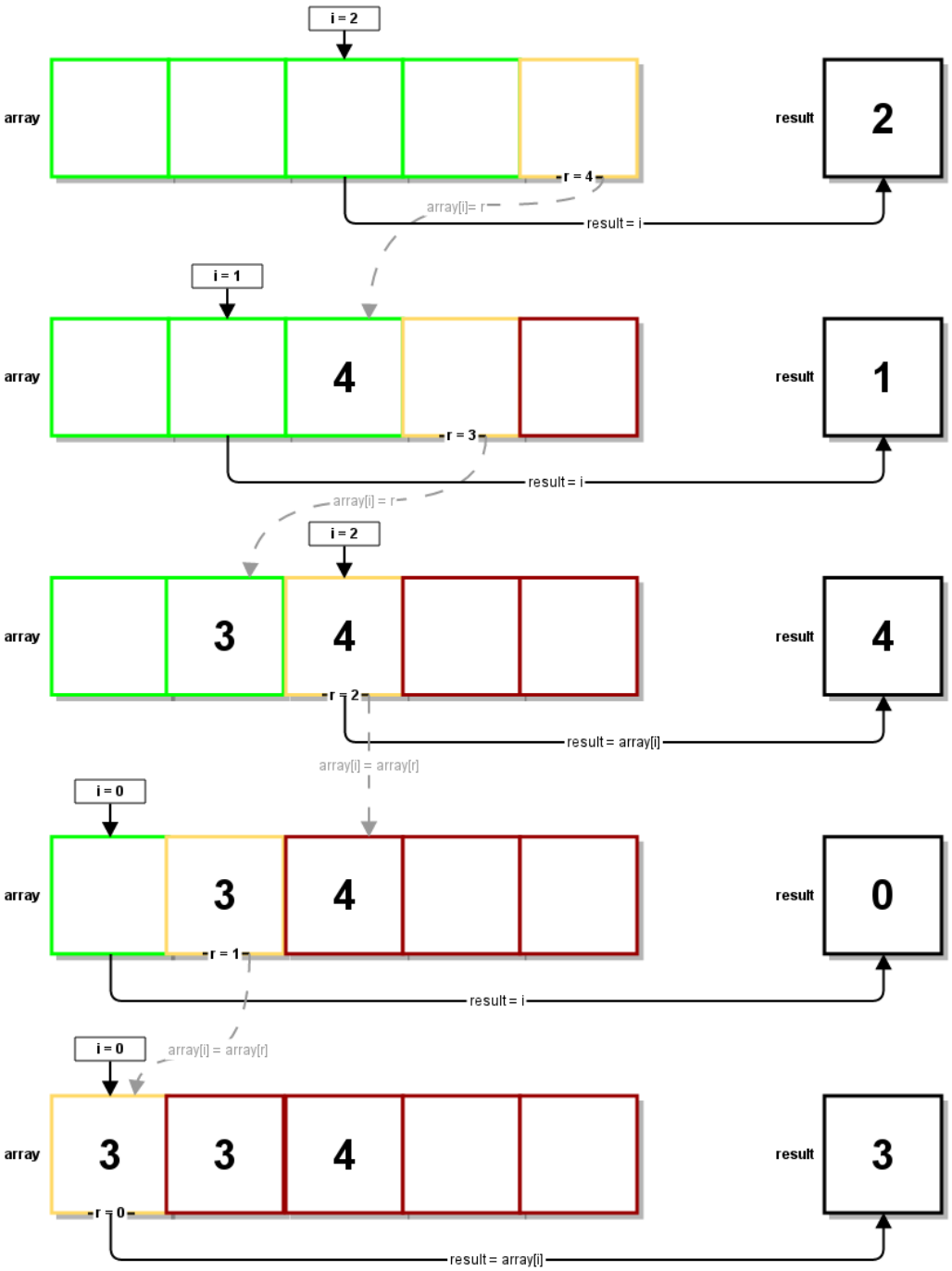
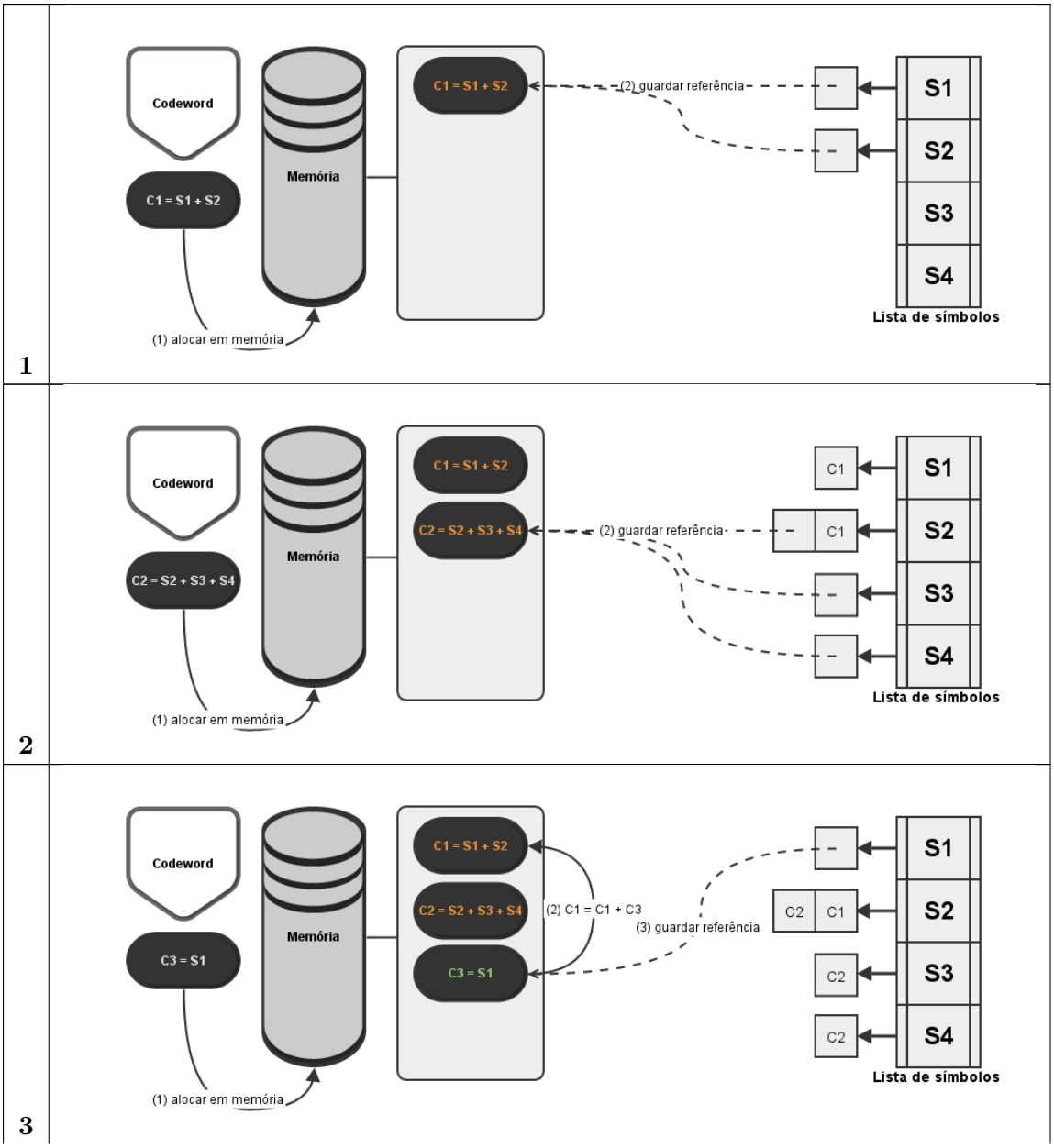
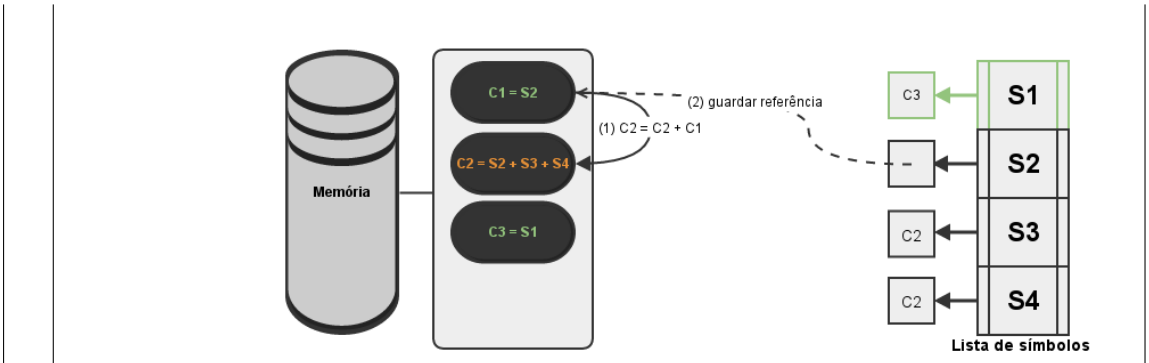


Figura 4.1: Diagrama exemplificativo do algoritmo desenvolvido de geração de números aleatórios sem repetição.

descodificadas, serão também referenciadas pelos símbolos correspondentes, utilizando uma variável específica que quando diferente do valo nulo, indica que o símbolo está descodificado e o seu valor é a referência para a *Codeword* respetiva (ver tabela 4.1).

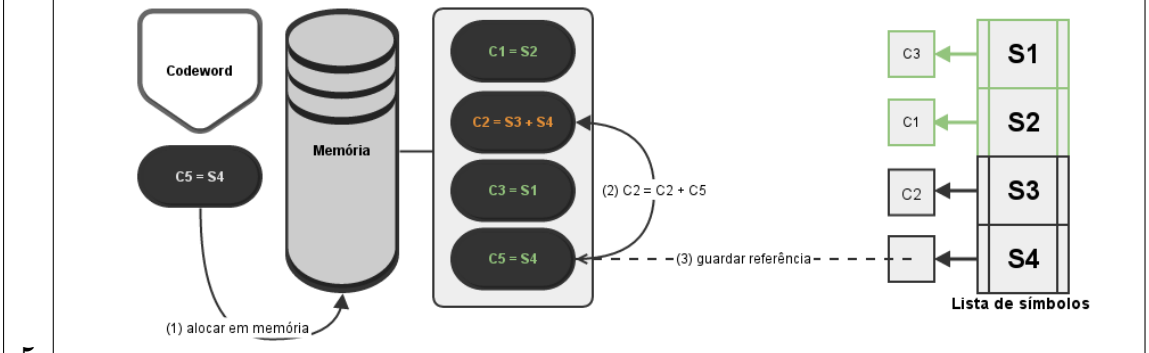


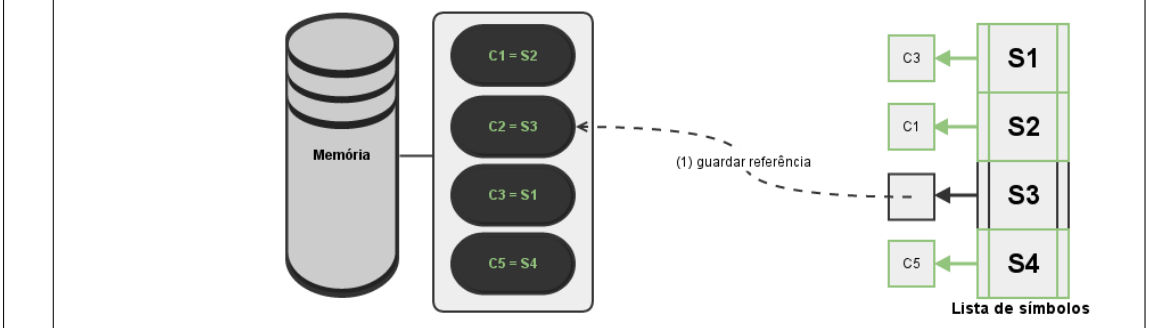

Memória
C1 = S2
C2 = S2 + S3 + S4
C3 = S1
(2) guardar referência
(1) C2 = C2 + C1
C3
–
C2
C2
S1
S2
S3
S4
Lista de símbolos

4


Codeword
C4 = S2
(1) eliminar
Memória
C1 = S2
C2 = S3 + S4
C3 = S1
C3
C1
C2
C2
S1
S2
S3
S4
Lista de símbolos

5


Codeword
C5 = S4
(1) alocar em memória
Memória
C1 = S2
C2 = S3 + S4
C3 = S1
C5 = S4
(2) C2 = C2 + C5
(3) guardar referência
C3
C1
C2
–
S1
S2
S3
S4
Lista de símbolos


Memória
C1 = S2
C2 = S3
C3 = S1
C5 = S4
(1) guardar referência
C3
C1
–
C5
S1
S2
S3
S4
Lista de símbolos

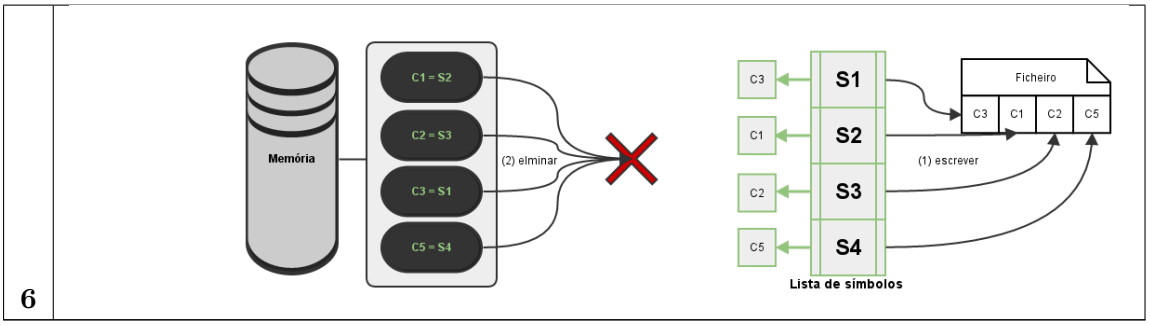


Tabela 4.1: Diagrama exemplificativo do processo de descodificação utilizando o algoritmo desenvolvido.

4.3 Aplicações

Nesta dissertação, foram desenvolvidas duas aplicações (fbp-client e fbp-server, ver secções 4.3.2 e 4.3.3 respetivamente) de forma a permitir a difusão e receção de um ficheiro codificado usando códigos Fountain, através de um canal de comunicação sujeito a perda de pacotes e sem necessidade de *feedback* por parte do(s) recetor(es), útil em situações em que o *feedback* por parte dos recetores não é possível ou indesejável.

Assim, para além das aplicações, foi criado um novo protocolo de comunicação, FBP (File Broadcast Protocol, ver secção 4.3.1) , possibilitando transmissão em difusão, sobre a camada de ligação de dados (L2), de ficheiros codificados com códigos Fountain.

4.3.1 Comunicação

Como descrito na secção 3.2, a comunicação entre as aplicações é feita sobre a camada de ligação de dados (L2), sem a utilização de nenhum protocolo de transporte (L4) nem nenhum protocolo de rede (L3), sendo a camada de ligação de dados responsável por garantir a integridade dos pacotes e a utilização de códigos Fountain como forma de suprimir a perda de pacotes sem necessidade de *feedback* [21]. Assim, na comunicação entre as aplicações, são utilizados *sockets* (L2) pelo servidor e cliente para o envio e receção de tramas respetivamente.

Como não existe *feedback* por parte dos recetores, apenas a aplicação responsável pela transmissão do ficheiro (fbp-server) transmite pacotes na rede, limitando-se as aplicações dos recetores (fbp-client) a receber os pacotes transmitidos.

Os pacotes transmitido pelo servidor são pacotes FBP (ver secção 3.2.1), sendo transmitida uma *Codeword* por pacote. Em cada pacote é também incluída informação relevante sobre a *Codewords* e mensagem original, como o número de símbolos que compõem a mensagem, a semente utilizada na geração dos índices da *Codeword* e o grau da *Codeword*. Esta informação é vital para que os clientes consigam descodificar as *Codewords* e reconstruir o ficheiro transmitido pelo servidor.

4.3.2 fbp-client

A aplicação fbp-client processa apenas os pacotes do tipo 0x1986 recebidos por uma determinada interface de rede. Todos os pacotes FBP recebidos pela interface de rede definida no momento de iniciação da aplicação são processados por esta aplicação.

Iniciada a aplicação, o primeiro pacote FBP recebido é utilizado para instanciar a classe Decoder (ver secção 4.2.3) com o valor de k passado no pacote, sendo o valor do tamanho das *Codewords* fixo (1488 bytes). De seguida é iniciado o processo de descodificação e adicionada a *Codeword* recebida ao processo.

Até que a aplicação seja interrompida ou até que o processo de descodificação obtenha k símbolos originais, a aplicação entra num ciclo contínuo a receber pacotes FBP e a passar as *Codewords* recebidas para a classe Decoder.

Após a conclusão do ciclo, e caso a aplicação não tenha sido interrompida, ou seja, tenham sido descodificados k símbolos, o ficheiro recebido é escrito numa determinada zona de memória ou num dispositivo de armazenamento de dados.

```
Modo de utilização: fbp-client -i <interface> -o <out_file>
-i Interface a utilizar na transferência
-o Ficheiro de escrita dos dados recebidos

Exemplo: fbp-client -i wlan0 -o /tmp/out.iso
```

4.3.3 fbp-server

A aplicação fbp-server envia constantemente pacotes do tipo 0x1986 (FBP) através de uma determinada interface de rede, ou seja, está constantemente a debitar pacotes FBP na rede através da interface definida no momento de iniciação da aplicação.

Iniciada a aplicação, esta começa por instanciar a classe Encoder (ver secção 4.2.2), com os dados de um determinado ficheiro. De seguida, esta entra num ciclo infinito onde é gerada uma *Codeword* e transmitida utilizando o protocolo FBP, a cada iteração. Este processo é executado infinitamente, até a aplicação ser terminada explicitamente.

```
Modo de utilização: fbp-server -i <interface> -f <file> [-d <MAC_dest>]
-i Interface a utilizar na transferência
-f Ficheiro a transmitir
-d (opcional) Permite definir uma transmissão unicast

Exemplo: fbp-server -i wlan1 -f file.iso
```

4.4 Scripts

Para auxiliar a configuração e execução global do sistema DETIboot, foram desenvolvidos *shell scripts* com diferentes funções, permitindo assim uma fácil utilização do DETIboot e minimizando possíveis erros de utilização/configuração do mesmo.

4.4.1 init-adhoc-client.sh

Script responsável pela configuração de uma interface de rede WiFi, para utilização no contexto DETIboot, utilizando a ferramenta *iwconfig* que, embora obsoleta, é compatível a maioria dos *drivers* para dispositivos de rede sem fios WiFi em sistemas Linux.

```
Modo de utilização: init-adhoc-client.sh -i <interface> -c <canal>
-i Interface a configurar
-c Canal da rede WiFi

Exemplo: init-adhoc-client.sh -i wlan0 -c 6
```

4.4.2 init-adhoc-server.sh

Script responsável pela configuração de uma interface de rede WiFi, para utilização no contexto DETIboot, utilizando a ferramenta *iw*. Esta aplicação embora mais avançada que a *iwconfig*, pode provocar problemas de compatibilidade com alguns *drivers* de dispositivos de rede sem fios WiFi mais antigos.

Este *script* é utilizado pelo servidor e tem como principal vantagem em relação ao *script* dos clientes (ver secção 4.4.1), definir o *basic rate* e *multicast rate* em 54 Mbits/s (máximo da norma 802.11g), permitindo assim aumentar a taxa de transmissão de pacotes em difusão na rede sem fios WiFi (ad-hoc).

```
Modo de utilização: init-adhoc-server.sh -i <interface> -c <canal>
-i Interface a configurar
-c Canal da rede WiFi

Exemplo: init-adhoc-server.sh -i wlan1 -c 6
```

4.4.3 DETIboot

O *script* DETIboot foi desenhado para correr num sistema de ficheiros Initramfs (ver secção 2.1.1), sendo este executado caso a opção "boot=DETIboot" esteja presente nos parâmetros de arranque do núcleo.

Este *script* começa por configurar uma interface de rede utilizando o *script* init-adhoc-cliente.sh (ver secção 4.4.1), de seguida é iniciada a transferência de uma imagem ISO utilizando a aplicação fbp-cliente (ver secção 4.3.2). Concluída a transferência, a imagem é montada numa pasta temporária e executado o ficheiro *startup* presente na mesma. Este ficheiro *startup* pode ser um qualquer tipo de ficheiro executável em ambiente Linux, tendo sido utilizado no desenvolvimento desta dissertação um ficheiro *shell script* contendo as instruções necessárias para o arranque do sistema operativo.

4.5 Integração

A integração final do cliente e servidor permite distribuir e arrancar um sistema operativo, utilizando redes sem fios WiFi, de forma eficiente. A rede é utilizada em modo ad-hoc e o sistema operativo transmitido em difusão pelo servidor, sem *feedback* sobre os pacotes recebidos por parte dos clientes, graças à codificação utilizada (códigos Fountain, ver secção

4.2), de forma a conseguir cumprir com os princípios básicos idealizados para o sucesso do DETIboot:

- Distribuição do sistema de ficheiros remota (melhor usabilidade) $\surd$
- Utilizadores sem permissões de escrita (melhor segurança) $\surd$
- Independência relativamente ao número de utilizadores (melhor desempenho) $\surd$

4.5.1 Cliente

O arranque de um sistema operativo utilizando o método de arranque DETIboot num sistema computacional é feito através de um núcleo Linux e respetivo Initramfs, sendo necessário que este Initramfs tenha incluído o método de arranque DETIboot, composto pelo próprio *script* de arranque DETIboot (ver secção 4.4.3), o *script* init-adhoc-client.sh para configurar a interface de rede (ver secção 4.4.1), a aplicação fbp-client para fazer a transferência do sistema operativo (ver secção 4.3.2) e o kexec para proceder à execução do sistema operativo transferido (ver secção 2.3). É necessário ainda incluir no Initramfs os *drivers wireless* próprios para o núcleo executado, para que seja possível utilizar dispositivos de rede sem fios WiFi durante a execução do método de arranque DETIboot, necessário para a transferência do sistema operativo. Para auxiliar a construção do Initramfs, foi utilizada a ferramenta *mkinitramfs*, disponível nas distribuições Ubuntu, que permite gerar um Initramfs para um determinado núcleo, a partir de ficheiros de configuração específicos, permitindo assim customizar o seu conteúdo.

A inicialização do núcleo e Initramfs é feita através de um *bootloader* Linux, tendo sido utilizado no desenvolvimento desta dissertação o *bootloader* GRUB (mas qualquer outro pode ser utilizado), sendo necessário que o *bootloader* passe alguns argumentos fundamentais ao núcleo no momento de arranque e existindo alguns argumentos extra opcionais:

- `"boot=DETIboot"`: Identifica qual o método de arranque que deve ser executado no Initramfs;
- `"if_wifi=<interface>"`: Identifica qual a interface de rede a utilizar pelo DETIboot;
- `"debug_mode"`: (Opcional) Execução do método de arranque DETIboot em modo debug;
- `"nomodeset"`: (Opcional) Indica ao núcleo para não carregar os *drivers* de vídeo, resolvendo problemas detetados durante a execução do kexec com algumas placas gráficas.

4.5.2 Servidor

O servidor do sistema DETIboot pode ser qualquer sistema computacional, desde que esteja a correr um sistema operativo Linux. Desta forma, é possível utilizar como servidor os mais diversos tipos de sistemas computacionais, desde os mais poderosos computacionalmente, até aos novos mini-PCs, embora o desempenho do mesmo possa sofrer variações.

Para colocar um sistema computacional a desempenhar a função de servidor do método de arranque DETIboot, basta executar o *script* init-adhoc-server.sh (ver secção 4.4.2) para configurar a interface de rede e após a conclusão deste, executar a aplicação fbp-server (ver secção 4.3.3) para transmitir um determinado ficheiro, sendo que em ambos deve ser utilizada a mesma interface de rede WiFi.

4.5.3 Imagem do Sistema Operativo

A imagem transmitida pelo servidor deve estar no formato ISO e conter pelo menos o ficheiro "startup", na sua raiz, com as instruções a executar pelos clientes para iniciarem o sistema operativo.

Para além do ficheiro "startup", é aconselhável que o ficheiro ISO transmitido contenha também um núcleo Linux e respetivo Initramfs com o sistema de ficheiros real do sistema operativo embutido, ou seja, incluir no Initramfs do núcleo o sistema de ficheiros final, de modo a não ser necessária a utilização de dispositivos de armazenamento de dados para alocar o sistema de ficheiros, ficando este armazenado em memória, aumentando assim o desempenho global do sistema, com a desvantagem de aumentar também o consumo de memória RAM dos clientes durante a execução do sistema operativo.

No desenvolvimento desta dissertação, foi utilizada uma imagem ISO composta pelo núcleo e Initramfs de uma distribuição *live* (Slax) e incluído neste Initramfs o sistema de ficheiros da distribuição. Optou-se pela distribuição Slax devido ao facto de esta estar de certa forma otimizada para correr apenas em memória RAM.

O ficheiro "startup" utilizado é composto por apenas uma instrução que consiste na execução do kexec (ver secção 2.3) utilizando o núcleo e Initramfs presentes no ficheiro ISO transmitido e os argumentos do núcleo, necessários para a correta execução do sistema operativo em questão.

```
#!/bin/sh
# ficheiro startup utilizado no desenvolvimento da dissertação

kexec -f ./vmlinuz --initrd=./initrd.img --append="slax.flags=xmode"
```

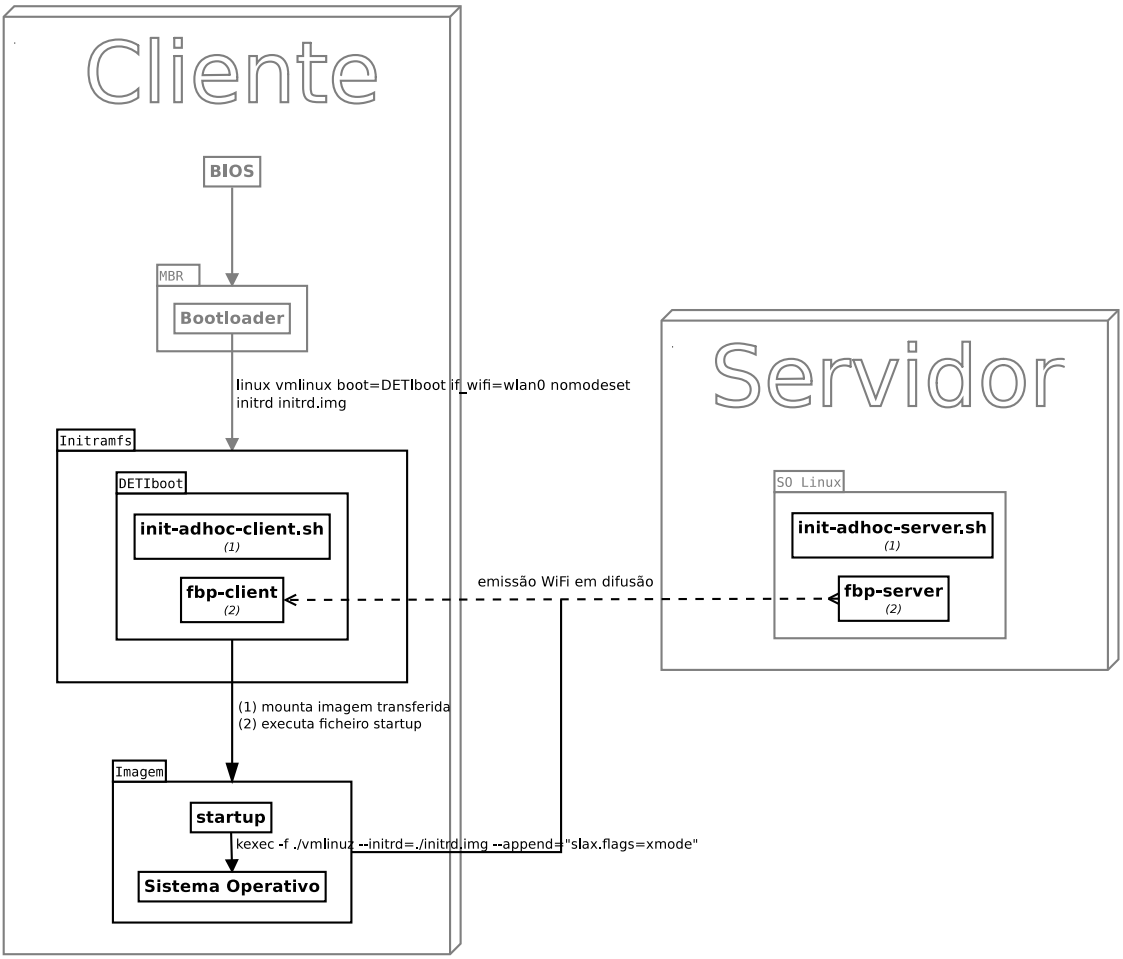


Figura 4.2: Diagrama ilustrativo da integração global do sistema DETIboot.

Capítulo 5

Avaliação do DETIboot

O método de arranque DETIboot pode ser avaliado fundamentalmente pelo desempenho em relação à distribuição e arranque de um sistema operativo, mais concretamente em relação ao tempo total despendido no arranque de um sistema operativo utilizando este método.

Assim, para avaliar a eficiência do sistema DETIboot, foi analisado o seu desempenho relativamente à comunicação sem fios WiFi e respetiva velocidade de transmissão (ver secção 5.1.1), processo de codificação e descodificação utilizando códigos Fountain (ver secção 5.1.2), arranque após a transferência do sistema operativo (ver secção 5.1.3) e o desempenho do sistema operativo transferido (ver secção 5.1.4).

Foram também identificadas algumas limitações presentes no método desenvolvido (ver secção 5.2), bem como realizados testes de campo relativamente à difusão de pacotes em redes sem fios WiFi e relativamente à execução do método de arranque DETIboot desenvolvido (ver secção 5.3).

5.1 Desempenho

O desempenho global do sistema DETIboot pode ser influenciado por diversos fatores e em diferentes fases do processo de arranque do sistema operativo. Sendo assim, a avaliação do desempenho do método de arranque DETIboot pode ser dividida por diferentes situações, cada uma com as suas particularidades, influenciando direta ou indiretamente o desempenho global do mesmo.

5.1.1 Comunicação

A comunicação em redes sem fios em geral é bastante sujeita a interferências e ruído, ficando assim a qualidade da comunicação dependente de fatores externos presentes no ambiente em que estas se encontram. Como o método de arranque utiliza este tipo de redes, o seu desempenho é afetado diretamente pelos mesmos fatores que influenciam o desempenho das comunicações sem fios, como por exemplo a presença de outras comunicações na mesma frequência de comunicação, objetos presentes na área abrangida (variando a influência consoante a sua composição), entre outros.

Em termos do perímetro de alcance do servidor, este está diretamente ligado à capacidade de alcance do dispositivo de rede sem fios WiFi utilizado pelo mesmo na transmissão do sistema operativo e, embora o alcance possa variar consoante o dispositivo de rede sem fios

WiFi utilizado, os clientes mais próximos do servidor terão sempre melhor desempenho na transferência do sistema operativo, pois consoante a distância aumenta, a relação sinal-ruído piora e por esse motivo a taxa de erros aumenta, originando que o número de tramas perdidas aumente.

Devido aos fatores mencionados, não foi possível encontrar um valor preciso para a taxa de transmissão de dados do sistema DETIboot, no entanto verificou-se que esta varia entre os 30 Mbits/s e 40 Mbits/s de dados úteis, ou seja, dados consumidos pelas aplicações desenvolvidas, descartando os dados relativos às camadas de rede inferiores nesta contabilização.

5.1.2 Códigos Fountain

Os processos de codificação e descodificação dos códigos Fountain exigem algum esforço computacional, podendo estes influenciar a taxa de transferência da imagem do sistema operativo.

De forma a minimizar este problema, foram utilizadas instruções vectorizadas, caso o processador do sistema computacional em questão as suporte, de forma a melhorar o desempenho das operações XOR realizadas, diminuindo assim o tempo despendido nesta tarefa.

Na aplicação fbp-client (ver secção 4.3.2), utilizada pelos clientes para receberem o sistema operativo, os processos de receção de pacotes e descodificação dos mesmos foram separados em *threads* independentes, permitindo assim receber novos pacotes mesmo que o processo de descodificação de uma determinada *Codeword* recebida ainda não esteja concluído, minimizando a influência do processo de descodificação na taxa de transferência.

Na aplicação fbp-server (ver secção 4.3.3), utilizada pelo servidor para transmitir a imagem do sistema operativo, a solução anterior não é aplicável, visto que o processo de transmissão é dependente do processo de codificação, ou seja, para transmitir um pacote é necessário que primeiro seja codificada uma *Codeword*. Assim, a capacidade de processamento do sistema computacional utilizado como servidor, tem mais influência na taxa de transferência do sistema operativo, e consequentemente no desempenho global do método de arranque DETIboot, visto que o desempenho do servidor influencia o desempenho dos clientes. Ao contrário do servidor, o desempenho de cada cliente não influencia o desempenho do servidor ou de qualquer outro cliente, visto estes terem um comportamento passivo, ou seja, limitam-se a escutar as tramas difundidas pelo servidor na rede sem emitirem qualquer trama para a mesma.

Outro fator que influencia diretamente o desempenho global do método de arranque DETIboot, é a escolha do valor de "c", respetivo à distribuição do grau (*robust soliton*, ver secção 2.5.1) utilizado na geração de *Codewords*. Este valor permite manipular a percentagem de *Codewords* de grau 1 (símbolos) durante o processo de geração de *Codewords*, influenciando assim o número de *Codewords* necessárias pelos clientes para conseguirem descodificar totalmente o ficheiro transmitido, bem como a memória utilizada pela aplicação.

De forma a encontrar um valor de "c" que permita obter um melhor desempenho do método de arranque DETIboot, foram analisados resultados dos 3 valores de "c" mais aconselháveis. Assim, as figuras 5.1, 5.2 e 5.3 demonstram os resultados obtidos das simulações realizadas referentes à transmissão de uma imagem com $k = 10000$, com diferentes valores de "c", onde os histogramas representam o número de *Codewords* utilizadas na descodificação total da imagem em 2000 experiências e os gráficos representam o número de *Codewords* armazenadas e descodificadas ao longo de uma descodificação da imagem transmitida.

Com estes resultados, foi possível verificar que a opção por um valor de "c" igual a 0.03 é a mais indicada devido a oferecer a melhor relação entre a média de *Codewords* necessária,

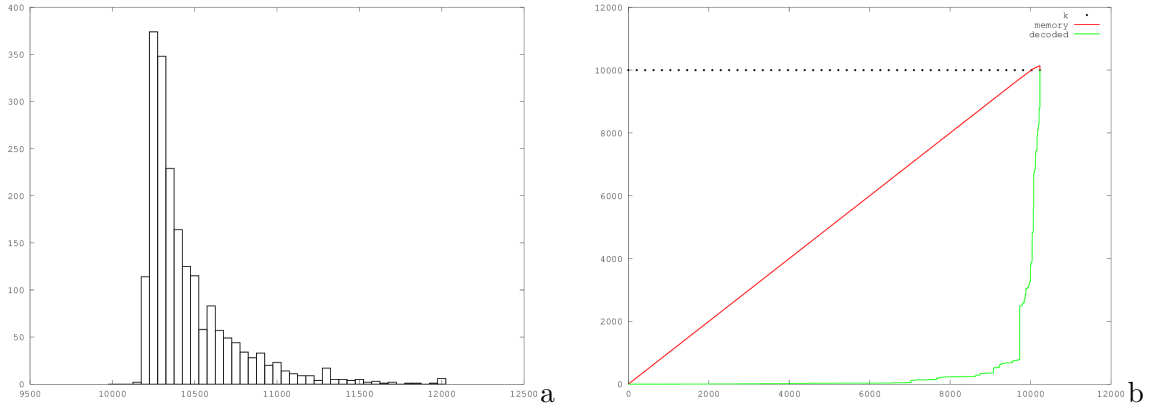


Figura 5.1: (a) Histograma representativo do número de *Codewords* necessárias para a descodificação completa da imagem transmitida e gráfico ilustrativo da utilização de memória durante uma descodificação (b), com $c = 0.01$

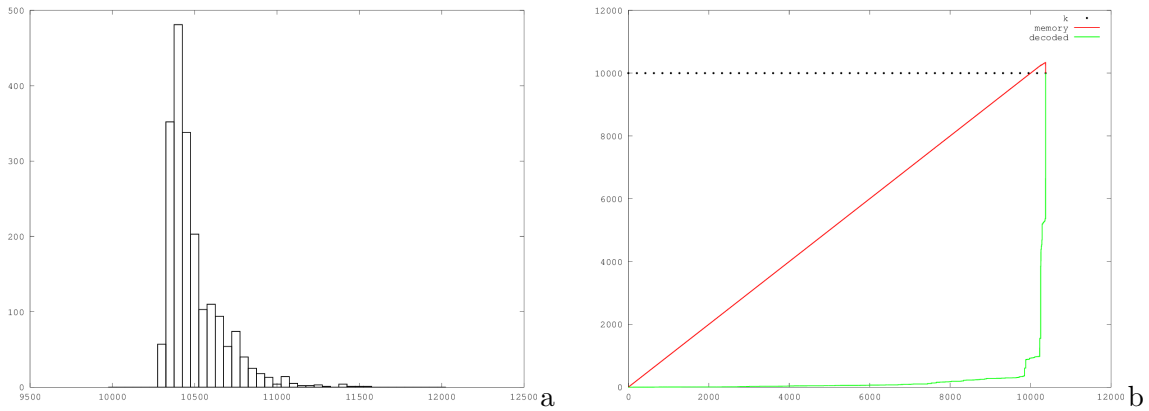


Figura 5.2: (a) Histograma representativo do número de *Codewords* necessárias para a descodificação completa da imagem transmitida e gráfico ilustrativo da utilização de memória durante uma descodificação (b), com $c = 0.03$

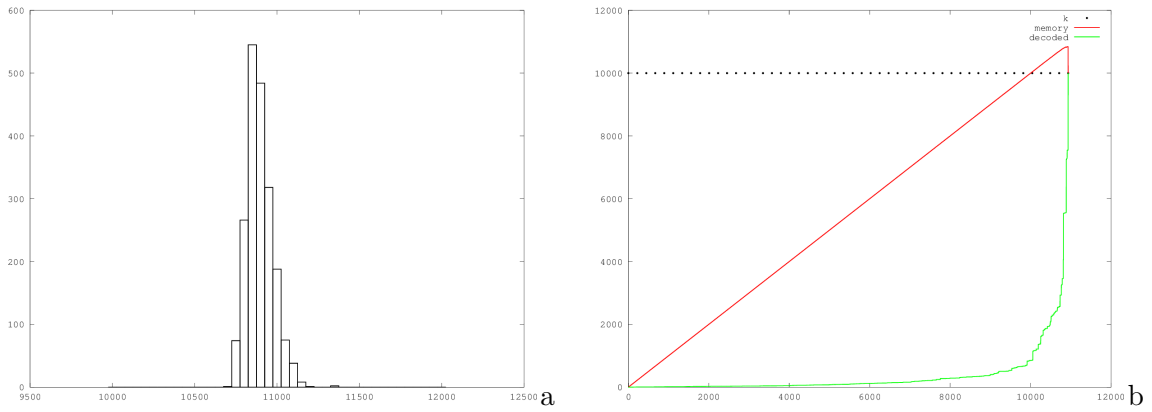


Figura 5.3: (a) Histograma representativo do número de *Codewords* necessárias para a descodificação completa da imagem transmitida e gráfico ilustrativo da utilização de memória durante uma descodificação (b), com $c = 0.1$

variação de *Codewords* necessárias (desvio padrão) e memória ocupada pela aplicação recetora, ou seja, permite que a descodificação total da imagem transmitida não varie muito entre os clientes (desvio padrão) tornando a transferência o mais imparcial possível, sejam necessárias menos *Codewords* quanto possível de modo a diminuir o tempo de descodificação (média), e diminuir o consumo de memória RAM que pode ser bastante vantajoso em máquinas com pouca capacidade da mesma.

5.1.3 Arranque

O tempo de arranque do sistema operativo é assim afetado pelos vários fatores mencionados anteriormente em relação à transferência deste, sendo o desempenho do sistema na fase posterior à transferência dependente da capacidade do sistema computacional em questão.

Após a transferência do sistema operativo, a inicialização deste é similar ao arranque normal de um sistema operativo Linux, tornando assim o desempenho desta fase idêntica ao método normal de arranque, tendo como principais fatores de influência a capacidade de processamento do sistema computacional utilizado, bem como a configuração do sistema operativo transferido. No entanto, nesta fase, o método de arranque DETIboot consegue ter melhor desempenho em relação ao método normal, visto que o sistema de ficheiros encontra-se em memória RAM, ao contrário do método normal em que este se encontra num dispositivo de armazenamento de dados, logo o método DETIboot consegue assim melhores velocidades nas operações de leitura e escrita no sistema de ficheiros em relação ao método normal.

Devido aos diversos fatores já mencionados, não é possível estabelecer um valor exato para o tempo total despendido no arranque de um sistema operativo utilizando o método de arranque DETIboot, como em qualquer outro método de arranque. No entanto comprovou-se que o arranque de um sistema operativo de aproximadamente 400 MB, utilizando o método DETIboot demora cerca de 2 a 3 minutos.

5.1.4 Sistema Operativo

O desempenho do sistema operativo transferido pode variar consoante a distribuição utilizada e as configurações feitas na mesma.

Nesta dissertação foi utilizada a distribuição Slax, devido a esta estar de certa forma preparada para correr unicamente em memória, sem recurso a nenhum dispositivo de armazenamento de dados, consumindo pouco mais de 700 MB de memória RAM após todo o processo de arranque estar concluído.

Embora a utilização da memória RAM para armazenar o sistema de ficheiros aumente significativamente o consumo desta, o desempenho global das operações de leitura e escrita no sistema de ficheiros também melhoram, devido à maior velocidade de acesso à memória em relação a um dispositivo de armazenamento de dados.

5.2 Limitações

Sendo a comunicação feita utilizando redes sem fios WiFi, a utilização do sistema DETIboot é limitado geograficamente, estando o seu alcance dependente do dispositivo WiFi utilizado pelo servidor, como descrito anteriormente no capítulo 3.

Para além da limitação geográfica, os clientes do sistema DETIboot necessitam de ter capacidade de memória RAM suficiente para o armazenamento e arranque do sistema operativo transmitido, ou seja, os clientes necessitam de ter no mínimo memória RAM disponível para armazenar o núcleo Linux e o Initramfs transmitidos, mais o tamanho ocupado pelo mesmo Initramfs descomprimido, pois durante a fase de inicialização do núcleo, o Initramfs é descomprimido por este em memória, e só após a finalização deste processo é que o Initramfs comprimido é descartado. Em valores concretos, desprezando a memória despendida no armazenamento do núcleo (30-40 *MB*), caso o Initramfs ocupe em memória 700 *MB* comprimido e 1500 *MB* descomprimido, é necessário que os sistemas computacionais dos clientes tenham no mínimo $700MB + 15000MB = 2200MB(2.2GB)$.

O desenvolvimento do método de arranque DETIboot ficou também limitado a clientes com sistemas computacionais de 64 *bits*, visto permitir melhor desempenho no processo de descodificação das *Codewords*, e a execução deste em sistemas computacionais Apple também não é suportada devido a características de arranque específicas dos mesmos. No entanto estas limitações poderiam ser resolvidas com a criação de diferentes versões para os vários sistemas, permitindo assim o suporte para máquinas de 32 *bits* e Apple.

Em relação ao servidor, este pode ser qualquer tipo de sistema computacional, no entanto nesta dissertação apenas foram desenvolvidas aplicações para sistemas operativos Linux. Um aspeto importante do servidor é que caso este não tenha memória RAM livre suficiente para armazenar o ficheiro transmitido, o processo de geração de *Codewords* irá criar uma atividade de paginação muito intensa o que causará um aumento significativo do tempo despendido nas operações de codificação, diminuindo assim drasticamente a velocidade de transmissão da imagem do sistema operativo.

Outro fator que pode limitar a velocidade de transmissão do servidor é a interface WiFi utilizada que, devido à utilização da ferramenta *iw* para definir o *basic rate* e *multicast*, é necessário que os *drivers* desta suportem a API nl80211, caso contrário a velocidade de transmissão de pacotes *broadcast* fica limitada a 1 *Mb/s*.

5.3 Testes de campo

No âmbito desta dissertação, foram realizados 2 testes de campo distintos, embora com os mesmos utilizadores e no mesmo ambiente, de forma a obter informações relevantes para a avaliação global do método de arranque DETIboot.

Os testes realizados tinham como objetivo avaliar a difusão de tramas em redes sem fios WiFi, e o desempenho global do método de arranque DETIboot.

Assim, as avaliações feitas foram separadas em 2 testes, um em relação à comunicação sem fios e outra em relação ao desempenho do sistema DETIboot.

Ambos os testes foram realizados no mesmo ambiente (sala de aula) e com os mesmos utilizadores (corpo académico convidado), com a distribuição representada na figura 5.4.

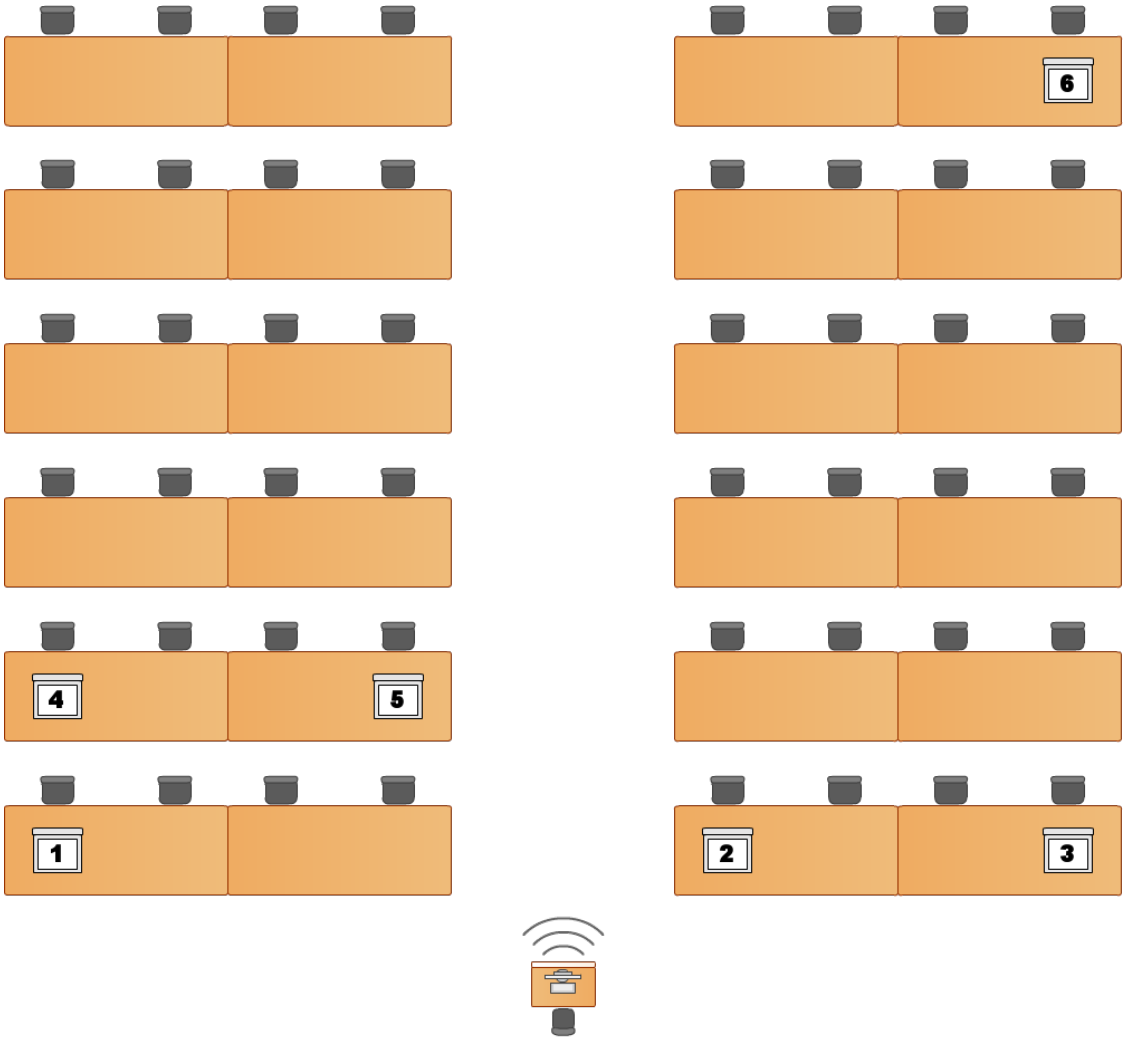


Figura 5.4: Diagrama ilustrativo da disposição dos utilizadores durante os testes realizados.

5.3.1 Dimensão ótima das tramas

Para se conseguir ter uma melhor perceção da difusão de pacotes em redes sem fios WiFi, foi feito um teste de modo a avaliar o desempenho desta, contabilizando a taxa de débito de dados do servidor e a perda de pacotes em cada recetor durante difusão de pacotes.

De forma a permitir esta contabilização, foi transmitido um ficheiro de aproximadamente 10 *MB* contendo dados aleatórios e incluindo no início de cada trama um campo de 32 *bits*, representando o índice de cada trama. Assim, os recetores conseguem identificar as perda de pacotes através da diferença dos índices das tramas recebidas.

No fim da transmissão, existe um espaço temporal para os recetores enviarem o número de perdas de pacotes para o servidor, permitindo assim uma fácil recolha dos resultados.

Para melhor avaliar o desempenho da transmissão, foram realizados 3 tipos de teste diferentes, onde cada tipo difere no comprimento das tramas utilizadas, sendo cada teste identificado por um número (índice), com as características presentes na tabela 5.1.

Ficheiro Transmitido $\simeq$ 10 MB		
Teste	**Comprimento das tramas (octetos)**	**Tramas transmitidas**
1	1500	7062
2	1000	10646
3	500	21621

Tabela 5.1: Índices dos testes realizados e respetivas características

Assim, durante o teste realizado com utilizadores, foram executados 3 vezes cada tipo de teste, tendo sido obtidos os resultados que se podem observar nas tabelas 5.2 e 5.3, em termos individuais e globais respetivamente.

Em relação à velocidade de transmissão também se observaram variações nos diferentes testes realizados, devido aos diferentes comprimentos das tramas utilizados, pois embora a taxa de transmissão de tramas se mantenha, a quantidade de informação em cada trama varia, alterando assim a taxa da transmissão de dados.

Assim, na tabela 5.4 são indicados os resultados obtidos nos testes realizados em relação ao tempo e velocidade de transmissão de um ficheiro de 10 *MB* para cada tipo de teste, 1 , 2 e 3.

Observando os resultados referidos, pode-se concluir que embora exista alguma variação em relação à perda de pacotes nos diferentes testes realizados, esta não é muito significativa, ao contrário da taxa de transmissão de dados onde se verifica bastante diferença entre os testes realizados, tornando assim a opção pelo comprimento máximo de trama (teste 1) a mais indicada visto se conseguir a taxa de transmissão de dados mais elevada, já que a diferença da perda de pacotes ser quase insignificante.

5.3.2 DETIboot

Após o teste à difusão de pacotes em redes sem fios WiFi, foi realizado um teste ao método de arranque desenvolvido no âmbito desta dissertação (DETIboot), sendo utilizado o comprimento de trama máximo, equivalente ao teste do tipo 1 anterior, devido a se ter comprovado que a diminuição do comprimento das tramas não era benéfico.

Este teste consistiu na execução integral do método de arranque DETIboot, tendo sido contabilizado o tempo despendido pelos sistemas computacionais dos utilizadores até terem o sistema operativo completamente inicializado e aparecer o ambiente de trabalho do SO.

Após o teste, foi feita uma recolha de informação junto dos utilizadores, respetiva ao tempo despendido pelo arranque do sistema operativo, utilizando o método de arranque DETIboot, e algumas características dos seus sistemas computacionais bem como eventuais problemas encontrados.

Os resultados obtidos neste teste conseguiram superar as expectativas, embora o número de utilizadores tivesse ficado aquém do que era esperado, pois verificou-se em alguns casos que o tempo de arranque foi inferior ao esperado (menos de dois minutos), para além de o tempo máximo registado ser de apenas pouco mais de dois minutos e meio. Registaram-se também dois problemas na execução do método de arranque DETIboot, no entanto o problema resultante da execução do método de arranque DETIboot num sistema computacional de 32 *bits* era esperado e o problema provocado por conflitos com a placa de vídeo num dos sistemas computacionais testados tem fácil resolução (ver secção 5.3.3 para mais detalhes). Na figura 5.5 são descritos os resultados obtidos, relativamente ao teste realizado.

5.3.3 Problemas

Nos teste realizados foram encontrados 2 problemas diferentes com sistemas computacionais distintos.

O primeiro problema encontrado já era conhecido, mas no entanto não esperado, tendo sido reportado pelo PC 4. A causa deste problema deveu-se ao sistema computacional em questão usar um processador com arquitetura de 32 *bits* e o método de arranque DETIboot utilizar um núcleo de 64 *bits* de forma a permitir acelerar o processo de descodificação, tendo assim este sistema computacional falhado ambos os testes. Embora este problema já fosse conhecido previamente, não era esperado encontrar utilizadores com sistemas computacionais com processadores de 32 *bits*, visto que já à algum tempo que esta tecnologia se tornou obsoleta.

O outro problema encontrado foi reportado pelo PC 6, e embora com problemas, foi possível contribuir nos testes realizados, visto que o problema ocorreu no momento de inicialização da interface gráfica do sistema operativo, permitindo assim disponibilizar informação relativamente à perda de pacotes e tempo de arranque, embora o sistema operativo tenha ficado inutilizável. Este problema foi causado pelos *drivers* de vídeo e reportado pelo utilizador como sendo um problema frequente no seu sistema computacional com várias distribuições Linux. Foi também possível identificar junto do utilizador uma solução para o problema, que passaria pela inclusão da opção *nomodeset* nos parâmetros de arranque do núcleo do sistema operativo, ou seja, incluir esta opção na execução do kexec, presente no ficheiro "startup" da imagem transmitida.

Assim, embora tenham ocorrido problemas na execução do método de arranque DETIboot, ambos os problemas reportados são solucionáveis, um através da utilização de um núcleo de 32 *bits* e outro através da inclusão da opção *nomodeset* nos parâmetros de arranque do núcleo do sistema operativo. No entanto, a experiência realizada pode não cobrir todos os tipos possíveis de sistemas computacionais devido ao baixo número de utilizadores presentes nos teste realizados.

PC 1				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	1,36%	1,36%	1,69%	1,97%
	1,97%			
	1,73%			
2	1,08%	0,84%	1,00%	1,08%
	0,84%			
	1,08%			
3	0,92%	0,92%	1,12%	1,28%
	1,17%			
	1,28%			

PC 2				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	-	4,80%	5,28%	5,75%
	5,75%			
	4,80%			
2	-	5,15%	5,27%	5,38%
	5,15%			
	5,38%			
3	-	4,19%	6,12%	8,05%
	4,19%			
	8,05%			

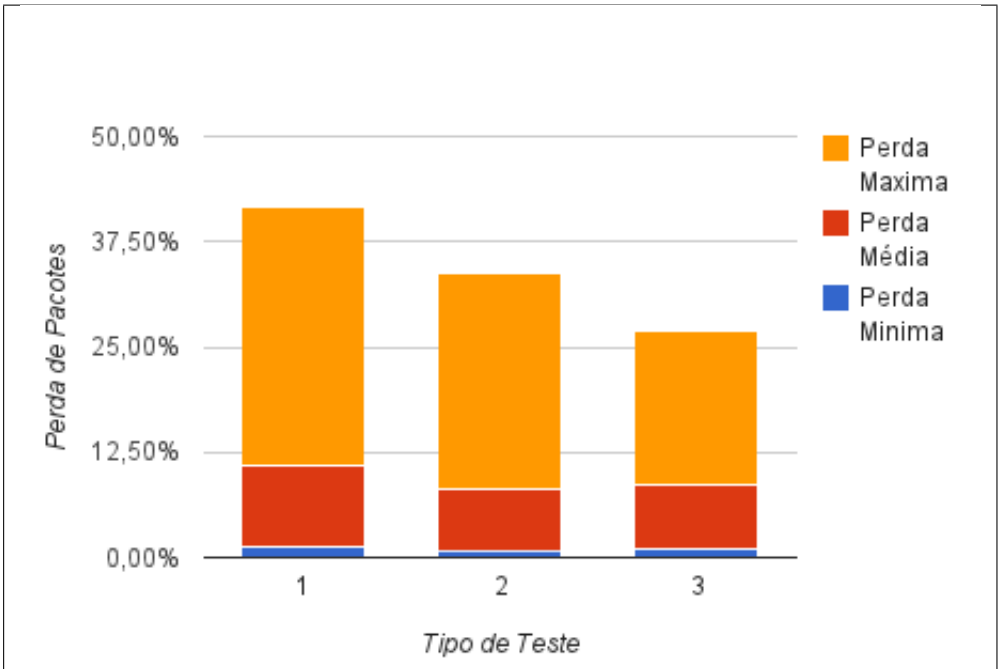
PC 3				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	13,45%	12,77%	17,34%	25,81%
	25,81%			
	12,77%			
2	7,25%	7,25%	13,92%	25,66%
	25,66%			
	8,84%			
3	9,23%	8,96%	10,99%	14,78%
	8,96%			
	14,78%			

PC 4				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	-	-	-	-
	-			
	-			
2	-	-	-	-
	-			
	-			
3	-	-	-	-
	-			
	-			

PC 5				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	6,40%	5,24%	5,79%	6,40%
	5,72%			
	5,24%			
2	8,00%	3,77%	6,14%	8,00%
	3,77%			
	6,65%			
3	7,37%	4,30%	6,54%	7,95%
	4,30%			
	7,95%			

PC 6				
Tipo de Teste	Perda de Pacotes	Perda Minima	Perda Média	Perda Maxima
1	-	5,99%	18,28%	30,56%
	30,56%			
	5,99%			
2	-	7,82%	9,64%	11,45%
	11,45%			
	7,82%			
3	-	9,67%	13,98%	18,28%
	18,28%			
	9,67%			

Tabela 5.2: Informação relativa à perda de pacotes individual de cada utilizador, nos testes realizados.



Tipo de Teste	Perda Mínima	Perda Média	Perda Máxima
1	1,36%	9,67%	30,56%
2	0,84%	7,19%	25,66%
3	0,92%	7,75%	18,28%

Tabela 5.3: Informação relativa à perda de pacotes global dos utilizadores, nos teste realizados.

Resultados (imagem $\simeq$ 10 MB)						
Teste	**Tempo de transmissão (s)**			**Taxa de transmissão (Mb/s)**		
	mínimo	**médio**	**máximo**	**mínimo**	**médio**	**máximo**
1 (1500 bytes)	2.53	2.73	2.88	29.11	30.87	33.22
2 (1000 bytes)	3.20	3.43	3.78	22.18	24.62	26.24
3 (500 bytes)	5.29	8.55	10.50	7.99	10.79	15.87

Tabela 5.4: Resultados obtidos pelo servidor, durante a realização dos testes de avaliação da difusão de tramas em redes WiFi

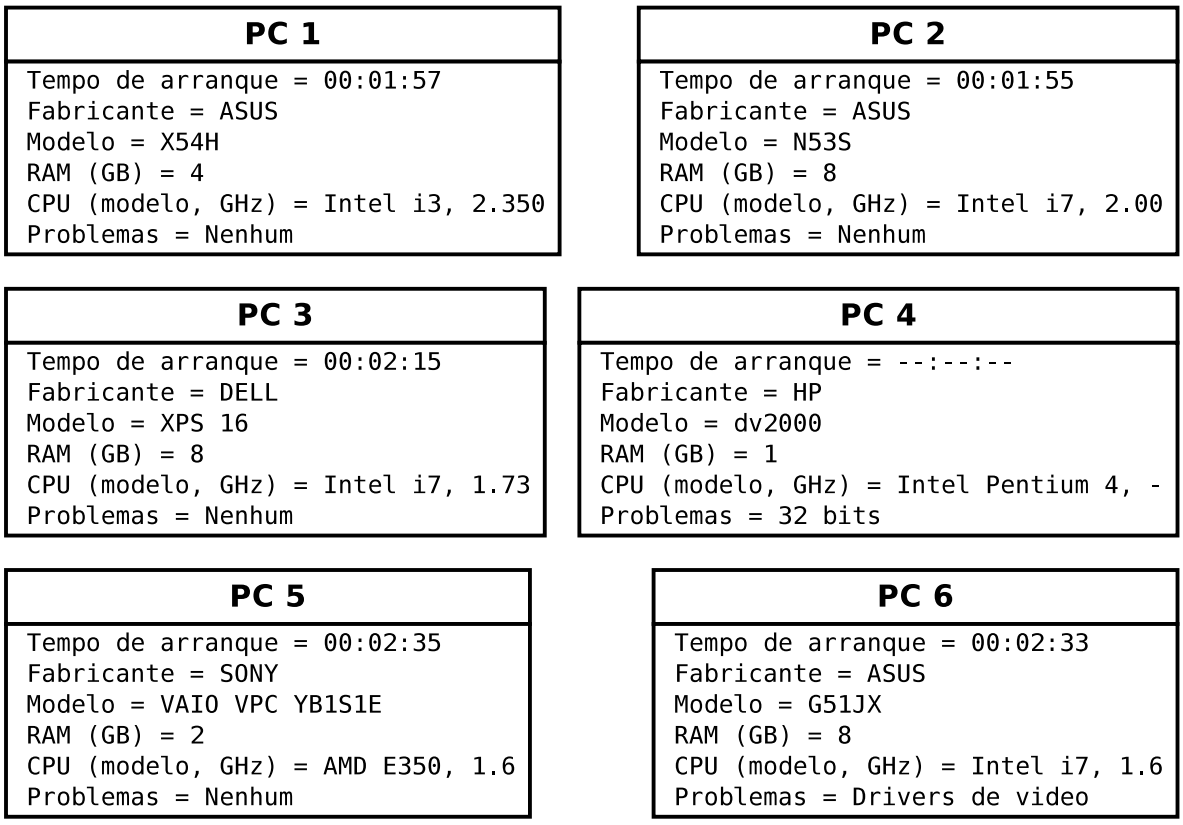


Figura 5.5: Informação relativa aos sistema computacionais utilizados e desempenho dos mesmos, no teste realizado ao método de arranque DETIboot.

Capítulo 6

Conclusão

O principal objetivo do DETIboot passava pelo desenvolvimento de uma alternativa viável aos computadores obsoletos disponibilizados pelas universidades, mais concretamente à possibilidade de utilização de computadores pessoais durante a realização de exames práticos.

Concluído o desenvolvimento, o DETIboot apresenta argumentos válidos para se tornar uma realidade num futuro próximo, dada a sua facilidade de utilização e o desempenho demonstrado, permitindo assim às universidades reduzir significativamente o número de computadores disponibilizados e assim poupar nos custos de manutenção e renovação destes.

Para além de resolver o problema a que foi proposto, o DETIboot apresenta uma forma inovadora de arranque nos sistemas operativos Linux, abrindo assim um infindável número de possibilidades de utilização do DETIboot. Para além da realização de exames práticos, por exemplo na realização de *workshops*, passa a ser possível distribuir um sistema operativo preparado para o mesmo por todos os participante, prescindindo das pré-configurações necessárias por parte dos utilizadores que normalmente criam problemas devido às diferentes configurações e variantes de sistemas operativos utilizados.

Durante o desenvolvimento do DETIboot foram analisadas várias formas de arranque de sistemas operativos Linux, de forma a encontrar um método de arranque que explore as falhas existentes, melhorando o desempenho e a usabilidade das soluções atuais no arranque de sistemas operativos Linux remotamente. O DETIboot, para além de conseguir cumprir esses objetivos, permite ainda que surjam novos casos de utilização, como o exemplo dos *workshops* referido anteriormente, tornando assim o trabalho realizado no âmbito desta dissertação bastante útil para a evolução do arranque de sistemas operativos remotamente.

Após o desenvolvimento e avaliação do método de arranque DETIboot, foi ainda realizada uma experiência com outra distribuição Linux (Slitaz), tendo-se observado melhorias de desempenho. A opção por esta distribuição deveu-se ao facto de o sistema de ficheiros desta ser já no formato Initramfs e possuir mecanismos de geração de imagens customizadas.

Em termos de formação, esta dissertação foi extremamente enriquecedora, visto complementar o conhecimento adquirido ao longo do percurso académico, mais concretamente nas áreas de sistemas operativos, redes de comunicação e *network coding*, bem como experiência em investigação científica.

Concluindo, considero o trabalho desenvolvido nesta dissertação de enorme utilidade, tanto em termos de contributo científico, como a nível de experiência pessoal visto ter conseguido contribuir com algo inovador e ter conseguido adquirir bastante conhecimento complementar durante a realização da mesma.

6.1 Trabalho futuro

Embora o resultado final do método de arranque DETIboot desenvolvido no âmbito desta dissertação possibilite já a distribuição e arranque de sistemas operativos Linux em computadores de terceiros, existem ainda algumas limitações que podem ser corrigidas (ver secção 5.2) e é possível que surjam eventuais problemas com uma maior utilização do mesmo, visto o teste realizado não ter tido a afluência desejada, podendo assim existir sistemas computacionais problemáticos com características diferentes dos testados. Assim, um aspeto importante no futuro será a realização de uma experiência de utilização do método de arranque DETIboot com um número alargado de utilizadores.

Outro aspeto que foi analisado mas não concretizado é a execução do método de arranque DETIboot em máquinas que não utilizem a BIOS na fase de inicialização do sistema computacional, como é o caso dos sistemas computacionais Apple e outros mais recentes, que usam o EFI/UEFI em detrimento da BIOS.

A distribuição Linux utilizada no DETIboot (Slax), embora otimizada para correr em memória, foi idealizada para ser carregada através de um dispositivo de armazenamento de dados e não através de memória RAM, o que pode causar duplicação de dados em memória, podendo assim o desempenho do DETIboot ser melhorado através do desenvolvido de uma distribuição Linux base, idealizada para o método de arranque DETIboot, de forma a melhorar o desempenho do mesmo.

Outros aspetos importantes que poderiam ser implementados com o desenvolvimento de uma distribuição específica para o método de arranque DETIboot, seriam a implementação de mecanismos de segurança, como a prevenção de execução do sistema operativo em máquinas virtuais, mecanismos para fácil gestão e configuração do sistema operativo, como a alteração das permissões dos utilizadores e criação dos mesmos ou até a inclusão de novas rotinas de arranque, como a inclusão de pastas remotas (NFS) no sistema operativo permitindo ter dados persistentes, entre outros.

Assim, o DETIboot para além de resolver o problema a que foi proposto, pode ainda evoluir no futuro, dentro do mesmo contexto em que foi desenvolvido, ou até ser explorado em outras vertentes.

Bibliografia

[1] Wikipedia contributors. Filesystem hierarchy standard, August 2012. Page Version ID: 506284680. URL: http://en.wikipedia.org/w/index.php?title=Filesystem_Hierarchy_Standard&oldid=506284680.

[2] Linux filesystem hierarchy. URL: http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/Linux-Filesystem-Hierarchy.html.

[3] Sistema de ficheiros, May 2013. Page Version ID: 35682383. URL: https://pt.wikipedia.org/w/index.php?title=Sistema_de_ficheiros&oldid=35682383.

[4] cpio, May 2013. Page Version ID: 554360967. URL: http://en.wikipedia.org/w/index.php?title=Cpio&oldid=554360967.

[5] initrd, April 2013. Page Version ID: 544196098. URL: https://en.wikipedia.org/w/index.php?title=Initrd&oldid=544196098.

[6] SquashFS, April 2013. Page Version ID: 548293581. URL: http://en.wikipedia.org/w/index.php?title=SquashFS&oldid=548293581.

[7] Linux startup scripts. URL: http://luv.asn.au/overheads/linux-startup.html.

[8] LiveCDCustomization - community ubuntu documentation. URL: https://help.ubuntu.com/community/LiveCDCustomization.

[9] Network file system, May 2013. Page Version ID: 556825855. URL: http://en.wikipedia.org/w/index.php?title=Network_File_System&oldid=556825855.

[10] Spencer Shepler, Mike Eisler, David Robinson, Brent Callaghan, Robert Thurlow, David Noveck, and Carl Beame. I-d tag:. URL: http://tools.ietf.org/html/rfc3530.

[11] diskless booting with PXE and NFS | pixelchaos.net. URL: http://www.pixelchaos.net/2009/02/15/diskless-booting-with-pxe-and-nfs/.

[12] DisklessUbuntuHowto - community ubuntu documentation. URL: https://help.ubuntu.com/community/DisklessUbuntuHowto.

[13] kexec, November 2012. Page Version ID: 509652774. URL: http://en.wikipedia.org/w/index.php?title=Kexec&oldid=509652774.

[14] Reboot linux faster using kexec, May 2004. URL: http://www.ibm.com/developerworks/linux/library/l-kexec/index.html.

[15] Rede sem fio, May 2013. Page Version ID: 35625866. URL: http://pt.wikipedia.org/w/index.php?title=Rede_sem_fio&oldid=35625866.

[16] Weiwei Fang, Feng Liu, Zhen Liu, Lei Shu, and S. Nishio. Reliable broadcast transmission in wireless networks based on network coding. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 555–559, 2011. doi:10.1109/INFCOMW.2011.5928875.

[17] D. J C MacKay. Fountain codes. *Communications, IEE Proceedings-*, 152(6):1062–1068, 2005. doi:10.1049/ip-com:20050237.

[18] Pedro Miguel Boto Saraiva. NFS fountain : sistema de ficheiros distribuído com códigos fountain, 2009. Mestrado em Engenharia de Computadores e Telemática. URL: http://ria.ua.pt/handle/10773/2158.

[19] M. Luby, M. Watson, T. Gasiba, T. Stockhammer, and W. Xu. Raptor codes for reliable download delivery in wireless broadcast systems. In *3rd IEEE Consumer Communications and Networking Conference, 2006. CCNC 2006*, volume 1, pages 192–197, 2006. doi:10.1109/CCNC.2006.1593014.

[20] Luby transform code, February 2013. Page Version ID: 517255326. URL: http://en.wikipedia.org/w/index.php?title=Luby_transform_code&oldid=517255326.

[21] Modelo OSI, May 2013. Page Version ID: 35912977. URL: http://pt.wikipedia.org/w/index.php?title=Modelo_OSI&oldid=35912977.